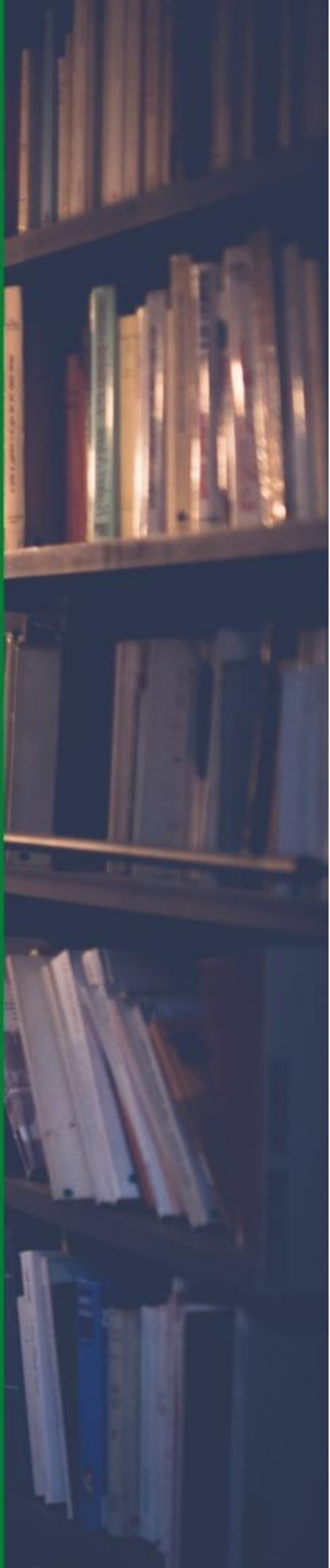
UMSIDA PRESS

BUKU AJAR

Teori Bahasa dan Automata



PENULIS

Sumarno
Hindarto
Suhendro Busono

BUKU AJAR
TEORI BAHASA DAN AUTOMATA

Oleh

Sumarno
Hindarto
Suhendro Busono

Diterbitkan oleh

Diterbitkan oleh
UMSIDA PRESS
Jl. Mojopahit 666 B Sidoarjo
ISBN: 978-623-464-031-1

BUKU AJAR
TEORI BAHASA DAN AUTOMATA

Penulis:
Sumarno
Hindarto
Suhendro Busono

ISBN :
978-623-464-031-1

Editor:
M.Tanzil Multazam,S.H,.M.Kn
Mahardika Darmawan Kusuma Wardana, M.Pd.

Copy Editor:
Wiwit Wahyu Wijayanti,S.H

Design Sampul dan Tata Letak:
Wiwit Wahyu Wijayanti,S.H

Penerbit:
UMSIDA Press

Redaksi
Universitas Muhammadiyah Sidoarjo
Jl. Mojopahit No 666B
Sidoarjo, Jawa Timur

Cetakan Pertama, September 2022

Prakata

Mata kuliah teori bahasa dan Automata merupakan mata kuliah yang sangat mendasar untuk mahasiswa yang ingin menpelajari berbagai ilmu Komputer atau informatika, khususnya bagi sarjana yang menekuni dunia teknologi informasi khususnya bidang desain computer, kecerdasan buatan dan analisis algoritma. Buku ini mengupas sejumlah pondasi dalam ilmu komputer dan beberapa contoh aplikasinya, buku ini mengkombinasikan antara teori, aplikasi serta contoh contoh soal dan penyelesaiannya, buku ini menunjukkan pemahaman terhadap bahasa–bahasa komputer, bahasa formal dan logika serta matematika komputer yang kadang-kadang sulit dimengerti dan dapat diterjemahkan ke dalam program dan aplikasinya secara mudah, oleh karena itu buku ini sangat berguna bagi mahasiswa teknologi informasi, teknik informatika dan komputer. Agar pembaca dapat menyerap materi yang disajikan dalam buku ini, dianjurkan untuk menguasai matematika diskrit, terutama pada materi teori himpunan, terori graph dan teori pohon urai.

Materi dalam buku ini disusun demikianrupa sehingga mempermudah pembaca untuk mempelajarinya. Oleh karena itu, sebaiknya bab-bab yang terdapat dalam buku ini dibaca dan dipelajari secara berurutan. Bab. 1 Memperkenalkan teori himpunan dan dasar Teori Graph, Bab 2 Pengantar Teori Bahasa, Aoutomata dan Aplikasinya, tentang string, Palindrome, Klenne star dan Language. Pada Bab 3. Ekpresi regular yaitu menjelaskan Bahasa regular, pengantar Tata bahasa Regular, Regular Expressions,dan Operasi-operasi Bahasa. Pada Bab 4 :Finite Automata (FA), Bab 5. Transition Graph, Bab 6. .Non Deterministik Finite Automata (NDFA), menjelaskan Transtion Table. Dan NDFA dengan ε-Move. Pada Bab 7. Finite Automata with Output menjelaskan Mesin Moore dan Mesin Mealy. Bab 8. Bahasa Regular, Bab 9. Context Free Grammer (CFG), Bab 10.Teori Pohon, menjelaskan Parse Tree.dan Ambiquity Grammer, Bab 11. Regular Grammer. Bab 12. Comsky Normal Form, Bab 13. Pushdawn Automata dan Bab 14. Turing Machine. Dengan selesainya penulisan buku ajar ini penulis mengucapkan terima kasih kepada semua pihak yang telah memberikan bahan-bahan tulisan baik langsung maupun tidak langsung.

Daftar Isi

Bab 1

Himpunan dan Graph

1.1 Himpunan

Kebanyakan kita memahami tentang himpunan dengan pengertian himpunan dan elemen-elemennya. atau "Set" dilihat sebagai kumpulan hal-hal sementara "elemen" dipandang sebagai hal-hal yang dimiliki set. Biasanya, himpunan didefinisikan pilihan tertentu yang dimiliki oleh elemen-elemennya. Sifat-sifat ini harus dijelaskan dengan baik, tanpa ambiguitas, sehingga selalu jelas apakah elemen tertentu termasuk dalam himpunan tertentu atau tidak. Menjadi "set" juga bisa pilihan elemen; jadi himpunan yang elemen-elemennya adalah himpunan ada. Misalnya, himpunan S dari semua tim di liga i tertentu. Unsur-unsur himpunan S adalah himpunan pemain hoki.

Mari kita perhatikan beberapa contoh entitas yang dapat kita anggap sebagai himpunan.

a. Misalkan T menyatakan himpunan semua garis lurus pada bidang kartesius. Misalnya, himpunan A = {(x, y) : y = 2x + 3} milik T sedangkan himpunan B = {1, 2, 3} tidak. Kita dengan mudah melihat bahwa T bukan elemen T karena T bukan garis dalam Cartesian pesawat terbang.
b. Misalkan U menyatakan himpunan semua himpunan yang mengandung banyak elemen tak terhingga. set ini terdefinisi dengan baik karena kita dapat dengan mudah membedakan elemen-elemen yang termasuk dalam U dari yang bukan milik U. Misalnya, himpunan bagian {−2, 0, 100} bukan elemen U karena hanya mengandung tiga elemen. Kami mengajukan pertanyaan: Apakah atur U menjadi elemen U? Untuk membantu menjawab pertanyaan ini, sebagaimana contoh

A0 = {0,1,2,3,4……}

A1 = {-1,0,1,2,3,4……}

A2 = {-2, -1, 0,1,2,3,4……}

An = { -n, -(n – 1), -(n- 2), ……., -2, -1 ,0 ,1, 2, 3, …….}

Setiap elemen dalam himpunan {A0,A1,A2,A2, . . .} milik U. Oleh karena itu, U mengandung banyak elemen yang tak terhingga. Kami menyimpulkan bahwa U adalah elemen dari U

c. Definisikan S sebagai himpunan semua "set yang bukan elemen dari dirinya sendiri". Sebagai contoh,himpunan T yang dijelaskan dalam contoh a) ada di S karena T bukan elemen itu sendiri.Himpunan U yang dijelaskan dalam contoh b) bukan milik S karena U adalah elemen dari diri.

Sekarang kita melihat lebih dekat pada tiga set yang dijelaskan di atas. Selain fakta bahwa itu adalah set yang sangat besar, tidak ada yang berlebihan tentang set T yang dijelaskan dalam contoh a). Di sisi lain, himpunan U yang dibahas dalam contoh b) juga tampaknya terdefinisi dengan baik karena himpunan yang tak terbatas dapat dengan mudah dibedakan dari yang tidak. Tetapi fakta bahwa himpunan ini adalah elemen dari dirinya sendiri membuatnya menjadi satu bertanya-tanya apakah kita harus mengizinkan set untuk memenuhi pilihan ini. (Brookshea 1989)

Di samping itu, sulit untuk mengungkapkan apa yang mungkin salah dengan set tersebut. Mari kita sekarang perhatikan baik-baik "set" yang dijelaskan dalam contoh c): Sebuah himpunan milik S hanya jika itu bukan milik dirinya sendiri. Kami bertanya-tanya apakah, seperti U dalam contoh b),himpunan S adalah elemen dari dirinya sendiri. Tetapi S tidak dapat menjadi milik S karena tidak ada elemen dalam S yang dapat milik dirinya sendiri. Jadi S bukan elemen itu sendiri. Kemudian, menurut definisi S, S akan kemudian menjadi elemen S. Ini tidak masuk akal. Jelas ada masalah dengan "set" yang dijelaskan dalam contoh c). Bahkan jika itu cukup mudah untuk mendeteksi kontradiksi yang mengikuti dari contoh c) secara khusus menentukan sumber kontradiksi ini bisa lebih sulit.

Pada Contoh c) dengan baik menggambarkan apa yang disebut paradoks. Seperti yang kami sebutkan sebelumnya, paradoks terdiri dari dua pernyataan kontradiktif yang keduanya secara logis mengalir dari apa yang dianggap sebagai konsep yang dipahami dengan baik dan didefinisikan dengan jelas. Pada kasus ini, pernyataan "S adalah unsur dari S" benar berarti pernyataan "S adalah elemen S" salah, dan sebaliknya. Bagi banyak orang, ini hanyalah permainan kata-kata; mungkin tampak cukup tidak berbahaya. Tapi bagi matematikawan ini bukan masalah sepele. Penemuan paradoks khusus ini oleh Bertrand Russell setara dengan mengungkap virus berbahaya yang terbengkalai di pusat sistem operasi sebuah komputer mikro. Itu tidak bisa diabaikan. Cara set didefinisikan bersama dengan sifat yang diterima secara universal membentuk dasar matematika modern. Paradoks seperti ini merusak kepercayaan yang kita miliki dalam matematika, suatu disiplin yang membanggakan kejernihan pemikirannya, sebuah disiplin yang melihat dirinya sebagai pencari kebenaran yang tak terbantahkan. Setelah cacat ini terungkap, penting untuk memahami alasannya. kami tidak melihat ini sebelumnya. Matematikawan juga bertanya-tanya apakah ada retakan lain di dasar matematika ada, membutuhkan perhatian segera.

1.1.1 Konsep Notasi

a. Jumlah istilah dan aksioma yang tidak terdefinisi harus sesedikit mungkin.
b. Biasanya, suatu aksioma tidak boleh dikurangkan secara logis dari aksioma lainnya. (Jika satu dapat dikurangkan dari yang lain, ini harus diungkapkan secara eksplisit.)
c. Kita harus dapat membuktikan dari aksioma dan konsep ini sebagian besar dari apa yang kita dianggap sebagai matematika yang menarik atau berguna. Seringkali, bagian dari logika alam semesta yang ditentukan dari aksioma sudah terbentuk sebelumnya, dalam arti bahwa aksioma diperkenalkan sehingga pernyataan matematika tertentu akan berubah keluar untuk menjadi kenyataan, tentu saja, bisa berubah menjadi "permainan berbahaya". Tetapi harus ada beberapa motivasi untuk memilih satu
d. Aksioma-aksioma ini tidak boleh mengarah pada paradoks apa pun. Suatu sistem aksiomatik yang mengandung kontradiksi dapat dimodifikasi menjadi kontradiksi di mana kontradiksi ini bekerja tidak terjadi atau beberapa aksioma dibuang begitu saja dan diganti dengan yang lain jika diperlukan.

Konsep primitif dalam teori kami. Akan ada tiga gagasan yang tidak terdefinisi dalam sistem aksioma. Contoh ekpresi

$$x \in A$$

harus dibaca sebagai "kelas x milik kelas A", atau "kelas x ada di kelas" A" atau "x adalah elemen dari A". Namun, tidak ada kelas yang dapat diwakili oleh huruf kecil huruf, x, kecuali diketahui bahwa x 2 B untuk beberapa kelas B. Kelas-kelas yang dapat diwakili oleh huruf kecil, katakanlah x, akan diberi nama khusus:

Jika kelas A sedemikian rupa sehingga A ϵ B untuk beberapa kelas B, maka kita akan mengacu pada kelas A sebagai "elemen"

1.1.2 Membangun Notasi Pembentuk Himpunan

{ x | syarat-syarat yang harus dipenuhi huruf x }

Dengan aturan:

a. Pada sisi kiri tanda '|' merupakan elemen himpunan
b. Pada simbul '|' dibaca dimana atau sedemikian hingga maka
c. Pada sisi kanan tanda'|' merupakan syarat keanggotaan himpunan
d. Pada Setiap tanda ',' adalah syarat ke-aggotaan dibaca sebagai dan

Sebagai contoh 1 : A : himpunan bilangan bulat positif lebih kecil dari 7

A = { x | x bilangan bulat positif lebih kecil dari 7}

atau A = { x | x P, x < 5 } yang ekivalen dengan A = {1, 2, 3, 4,5,6}

Contoh ditunjukkan

$$A = \{1, 2, 3\}.$$

Ini dibaca, "A adalah himpunan yang memuat elemen 1, 2 dan 3." Kita gunakan kurung kurawal "{, }" untuk mengapit elemen himpunan.

Beberapa notasi lagi:

$$a \in \{a, b, c\}.$$

Simbol "∈" dibaca "ada di" atau "adalah elemen dari." Jadi maksud di atas bahwa a adalah anggota himpunan yang memuat huruf a, b, dan c. Perhatikan bahwa ini merupakan pernyataan yang benar. Juga benar untuk mengatakan bahwa d tidak ada dalam himpunan itu

$$d \notin \{a, b, c\}.$$

Berhati-hatilah: kita menulis "x ϵ A" ketika kita ingin menyatakan bahwa salah satu dari anggota himpunan A adalah x. Misalnya, pertimbangkan himpunan,

$$A = \{1, b, \{x, y, z\}, \emptyset\}.$$

Ini adalah set yang aneh, tentu saja. Ini berisi empat elemen: nomor 1, huruf b, himpunan {x, y, z}, dan himpunan kosong (∅ {}, himpunan yang berisi tidak ada elemen). Apakah x di A? Jawabannya adalah tidak. Tak satu pun dari empat elemen di A adalah huruf x, jadi kita harus menyimpulkan bahwa x < A. Demikian pula, pertimbangkan himpunan B {1, b}. Meskipun unsur-unsur B adalah unsur-unsur A, kita tidak dapat mengatakan bahwa himpunan B adalah salah satu anggota dari A. Oleh karena itu B < A. (Segera kita lihat bahwa B adalah himpunan bagian dari A, tetapi ini berbeda dengan menjadi elemen dari A.) Kami telah menjelaskan himpunan di atas dengan mendaftar elemen-elemennya. Terkadang ini sulit dilakukan, terutama ketika ada banyak elemen di set (mungkin banyak tak terhingga). Misalnya, jika kita ingin A menjadi himpunan semua genap alami angka, bisa menulis,

$$A = \{0, 2, 4, 6, \ldots\},$$

tapi ini agak kurang tepat. Cara yang lebih baik adalah

$$A = \{x \in N : \exists n \in N(x \ 2n)\}.$$

Menguraikannya: "x ϵ N" berarti x ada di himpunan N (kumpulan natural bilangan, {0, 1, 2, . . .}), ":" dibaca "sehingga" dan "∃n N(x 2n)" dibaca

"ada n dalam bilangan asli di mana x adalah dua kali n" (di lain kata, x genap). Sedikit lebih mudah mungkin,

$$A \{x : x \text{ genap}\}.$$

Berikut adalah beberapa contoh lagi:

Jelaskan masing-masing himpunan berikut baik dengan kata-kata dan dengan mendaftar elemen yang cukup untuk melihat polanya.

1. $\{x : x + 3 \in N\}$.
2. $\{x \in N : x + 3 \in N\}$
3. $\{x : x \in N \vee -x \in N\}$.
4. $\{x : x \in N \wedge -x \in N\}$.

Jawaban

1. Ini adalah himpunan semua bilangan yang 3 lebih kecil dari bilangan nomor asli (yaitu, bahwa jika Anda menambahkan 3 ke mereka, Anda mendapatkan bilangan asli). Himpunan juga dapat ditulis sebagai {−3, -2, -1, 0, 1, 2, . . .} (perhatikan bahwa 0 adalah bilangan asli, jadi 3 ada di himpunan ini karena -3 + 3 = 0).
2. Ini adalah himpunan semua bilangan asli yang 3 kurang dari a bilangan asli. Jadi di sini kita hanya memiliki {0, 1, 2, 3 . . .}.
3. Ini adalah himpunan semua bilangan bulat (bilangan bulat positif dan negatif)bers, ditulis Z). Dengan kata lain, {. . . , -2, -1, 0, 1, 2, . . .}.
4. Di sini kita ingin semua bilangan x sedemikian rupa sehingga x dan -x adalah natural angka. Hanya ada satu: 0. Jadi kita memiliki set {0}.

1.1.3 Operasi Himpunan

Apakah mungkin untuk menambahkan dua Himpunan ? Tidak juga, namun ada yang serupa. Jika kita ingin menggabungkan dua himpunan untuk mendapatkan kumpulan objek yang berada di set, maka kita dapat mengambil gabungan dari dua set. Secara simbolis ditulis,

$$C = A \cup B,$$

dibaca, “C adalah gabungan dari A dan B,” berarti bahwa elemen-elemen C adalah tepat elemen yang merupakan elemen A atau elemen B (atau elemen keduanya). Misalnya, jika A = {1, 2, 3} dan B = {2, 3, 4}, maka A ∪ B, {1, 2, 3, 4}.

Operasi umum lainnya pada himpunan adalah persimpangan atau irisan ditulis,

$$C = A \cap B$$

dan katakan, "C adalah perpotongan A dan B," ketika elemen-elemen dalam C adalah tepatnya keduanya di A dan di B. Jadi

jika A ={1, 2, 3} dan

B = {2, 3, 4}, maka

$$A \cap B = \{2, 3\}.$$

Kita mungkin ingin berbicara tentang semua elemen yang tidak berada dalam himpunan tertentu. Kita katakan B adalah komplemen dari A, dan tulis,

$$B = \overline{A}$$

ketika B berisi setiap elemen yang tidak terkandung dalam A. Jadi, jika alam semesta kita adalah {1, 2, . . . , 9, 10}, dan A = {2, 3, 5, 7}, maka

$$\overline{A} = \{1, 4, 6, 8, 9, 10\}.$$

Tentu saja kita dapat melakukan lebih dari satu operasi pada satu waktu. Sebagai contoh, mempertimbangkan

$$A \cap \overline{B}.$$

Ini adalah himpunan semua elemen yang merupakan elemen dari A dan bukan elemen B. Apa yang telah kita lakukan? Kami sudah mulai dengan A dan menghapus semuanya elemen-elemen yang ada di B. Cara lain untuk menulis ini adalah perbedaan himpunan

$$A \cap \overline{B} = A \setminus B.$$

Penting untuk diingat bahwa operasi ini (penyatuan, irisan, komplemen, dan selisih) pada himpunan menghasilkan himpunan lain. Jangan bingung ini dengan simbol-simbol dari bagian sebelumnya (elemen dan subset dari). $A \cap B$ adalah himpunan, sedangkan $A \subseteq B$ benar atau salah. Ini adalah perbedaan yang sama seperti antara 3 + 2 (yang merupakan angka) dan $3 \leq 2$ (yang salah).

Misalkan

A ={1, 2, 3, 4, 5, 6},

B = {2, 4, 6},

C = {1, 2, 3} dan

D = {7, 8, 9}.

Jika alam semesta adalah U = {1, 2, . . . , 10},

tentukan:

1. $A \cup B$.
2. $A \cap B$.
3. $B \cap C$.
4. $A \cap D$.
5. $\overline{B \cup C}$.
6. $A \setminus B$.
7. $(D \cap \overline{C}) \cup \overline{A \cap B}$.

Jawaban

1. A ∪ B = {1, 2, 3, 4, 5, 6} A karena semua yang ada di B sudah ada di A
2. A ∩ B ={2, 4, 6} B karena semua yang ada di B ada di A.
3. B ∩ C = {2} sebagai satu-satunya elemen dari B dan C adalah 2.
4. A ∩ D ∅ karena A dan D tidak memiliki elemen yang sama.
5. = {5, 7, 8, 9, 10}. Pertama kita cari B C {1, 2, 3, 4, 6}, maka kami mengambil segala sesuatu yang tidak ada di set itu.
6. A \ B = {1, 3, 5} karena elemen 1, 3, dan 5 ada di A tetapi tidak di B. Ini sama dengan A ∩ B.
7. Maka

Himpunan berisi semua elemen yang baik dalam D tetapi tidak dalam C (yaitu, {7, 8, 9}), atau tidak dalam baik A dan B (yaitu, {1, 3, 5, 7, 8, 9, 10})

x merupakan elemen dari A ∪ B? Artinya x adalah elemen dari A atau x adalah elemen dari B (atau keduanya). Itu adalah,

$$x \in A \cup B \quad \Leftrightarrow \quad x \in A \vee x \in B.$$

Demikian pula,

$$x \in A \cap B \quad \Leftrightarrow \quad x \in A \wedge x \in B.$$

Juga,

$$x \in \overline{A} \quad \Leftrightarrow \quad \neg(x \in A).$$

yang menyatakan x adalah unsur komplemen A jika x bukan unsur A.

Ada satu cara lagi untuk menggabungkan set yang akan berguna bagi kita ;

kartesius product A × B

A × B {(a, b) : a ∈ A ∧ b ∈ B}.

Koordinat pertama berasal dari set pertama dan koordinat kedua berasal dari set kedua. Terkadang kita ingin mengambil produk Cartesian dari suatu himpunan A × A {(a, b) : a, b ∈ A}, (kita mungkin juga menulis A2 untuk himpunan ini). Perhatikan bahwa di A × A, kita masih menginginkan semua pasangan terurut, bukan hanya

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

yang koordinat pertama dan kedua sama. Kita juga bisa ambil produk dari 3 set atau lebih, dapatkan pesanan tiga kali lipat, atau empat kali lipat, dan segera.

Contoh

ditunjukkan A = {1, 2} dan B = {3, 4, 5}. Tentukan A × B dan A × A., Berapa banyak elemen yang Anda harapkan berada di B × B?

Jawaban

A × B = {(1, 3), (1, 4), (1, 5), (2, 3), (2, 4), (2, 5)}.

A × A = A2 {(1, 1), (1, 2), (2, 1), (2, 2)}

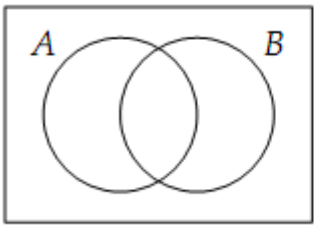
|B × B| = 9. Akan ada 3 pasang dengan koordinat pertama 3, tiga lebih banyak dengan koordinat pertama 4, dan tiga terakhir dengan koordinat pertama 5.

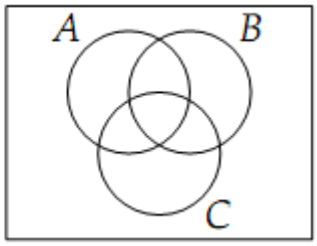
Di bawah ini ada beberapa notasi atau gambar simbol.

- ✓ 0 Himpunan kosong adalah himpunan yang tidak memiliki elemen.
- ✓ U Himpunan alam semesta adalah himpunan semua elemen.
- ✓ N Himpunan bilangan asli. Yaitu, N {0, 1, 2, 3 . . .}.
- ✓ Z Himpunan bilangan bulat. Artinya, Z{. . . , 2, 1, 0, 1, 2,3, . . }.
- ✓ Q Himpunan bilangan rasional.
- ✓ R Himpunan bilangan real.
- ✓ P(A) Himpunan pangkat dari sembarang himpunan A adalah himpunan semua himpunan bagian dari A.
- ✓ {,} Kami menggunakan kurung kurawal ini untuk mengapit elemen dari suatu himpunan. Jadi {1, 2, 3} adalah himpunan yang berisi 1, 2, dan 3.
- ✓ : {x : x > 2} adalah himpunan semua x sedemikian sehingga x lebih besar dari 2.
- ✓ 2 ϵ{1, 2, 3} menyatakan bahwa 2 adalah anggota dari himpunan {1, 2, 3}.ϵ 4 ϵ {1, 2, 3} karena 4 bukan anggota himpunan {1, 2, 3}.
- ✓ ⊆ A ⊆ B menyatakan bahwa A adalah himpunan bagian dari B: setiap elemen A juga sebuah elemen B
- ✓ ⊂ A ⊂ B menyatakan bahwa A adalah himpunan bagian sejati dari B: setiap elemen dari A juga merupakan elemen B, tetapi A tidak sama dengan B.
- ✓ ∩ A∩B adalah irisan dari A dan B: himpunan yang memuat semua elemen yang merupakan elemen dari A dan B.
- ✓ ∪ A ∪ B adalah gabungan dari A dan B: adalah himpunan yang memuat semua anggota yang merupakan elemen A atau B atau keduanya.
- ✓ × A × B adalah produk Cartesian dari A dan B: himpunan semua terurut pasangan (a, b) dengan a ϵ A dan b ϵ B.
- ✓ \ A \ B adalah A set-minus B: himpunan yang berisi semua elemen A yang bukan elemen B
- ✓ A Komplemen dari A adalah himpunan segala sesuatu yang bukan merupakan elemen A
- ✓ |A| |A| Kardinalitas (atau ukuran) dari A adalah jumlah elemen dalam A..

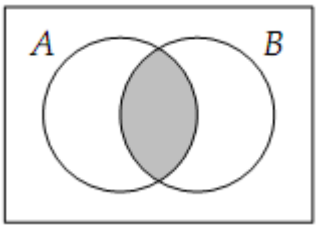
1.1.4 Diagram Venn

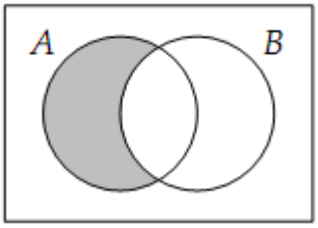
Ada alat visual yang sangat bagus yang dapat kita gunakan untuk merepresentasikan operasi pada set. Diagram Venn menampilkan himpunan sebagai lingkaran yang berpotongan. Kita bisa menaungi wilayah yang kita bicarakan ketika kita melakukan operasi. Kita juga dapat merepresentasikan kardinalitas suatu himpunan tertentu dengan memasukkan bilangan tersebut ke dalam wilayah yang sesuai.



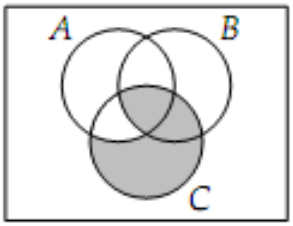


Setiap lingkaran mewakili satu set. Persegi panjang yang berisi lingkaran mewakili alam semesta. Untuk mewakili kombinasi set ini, kami menaungi wilayah yang sesuai. Sebagai contoh, kita dapat menggambar A ∩ B sebagai:





$A \cap \overline{B}$, or equivalently $A \setminus B$



$(B \cap C) \cup (C \cap \overline{A})$

Perhatikan bahwa daerah yang diarsir di atas juga dapat ditemukan caralain. Kita bisa memulai dengan semua C, lalu mengecualikan wilayah di mana C dan A tumpang tindih di luar B. Daerah itu adalah

$$(A \cap C) \cap \overline{B}.$$

Jadi diatas Diagram Venn juga mewakili

$$C \cap \left((A \cap C) \cap \overline{B}\right)$$

Jadi hanya menggunakan gambar, kami telah menentukan bahwa

$$(B \cap C) \cup (C \cap \overline{A}) = C \cap \overline{\left((A \cap C) \cap \overline{B}\right)}.$$

Contoh Kardinalitas

1. Cari kardinalitas dari A = {23, 24, . . . , 37, 38}.

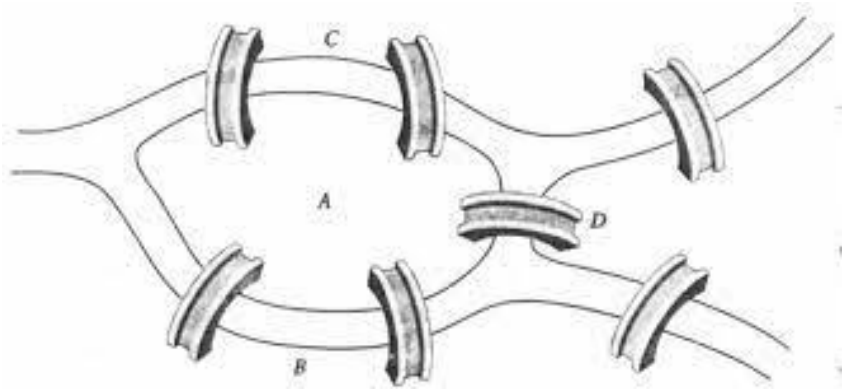
2. Tentukan kardinalitas dari B = {1, {2, 3, 4},0 }.

3. Jika C {1, 2, 3}, berapa kardinalitas dari P(C)?

1) Karena 38- 23 = 15, kita dapat menyimpulkan bahwa kardinalitas dari himpunannya adalah |A| 16 (Anda perlu menambahkan satu karena 23 disertakan).
2) 2. Di sini |B|= 3. Tiga elemen adalah 1, himpunan {2, 3, 4}, dan himpunan kosong.
3) Kami menuliskan elemen-elemen dari himpunan daya P(C) di atas, dan ada 8 elemen (masing-masing adalah himpunan). Jadi |P(C)|= 8. (Anda mungkin bertanya-tanya apakah ada hubungan antara |A| dan |P(A)| untuk semua set A. Ini adalah pertanyaan bagus yang akan kami.

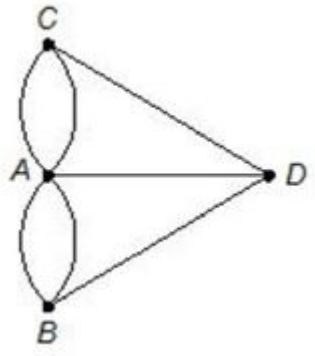
1.2 Graph

Pada zaman Euler, di kota Königsberg di Prusia, ada adalah sungai yang berisi dua pulau. Pulau-pulau itu terhubung ke tepi sungai dengan tujuh jembatan (seperti yang terlihat di bawah). Itu jembatan sangat indah, dan pada hari libur mereka, warga kota akan menghabiskan waktu berjalan di atas jembatan. Seiring berjalannya waktu,muncul pertanyaan: apakah mungkin untuk merencanakan jalan-jalan sehingga Anda menyeberang setiap jembatan sekali dan hanya sekali? Euler mampu menjawab pertanyaan ini.



Gambar 1.1. Jembatan Königsberg

Teori Graf adalah bidang matematika yang relatif baru, pertama kali dipelajari oleh matematikawan super terkenal Leonhard Euler pada tahun 1735. Sejak itu telah berkembang menjadi alat yang ampuh yang digunakan di hampir setiap cabang ilmu pengetahuan dan saat ini merupakan bidang penelitian matematika yang aktif. Masalah di atas, yang dikenal sebagai Tujuh Jembatan Königsberg, adalah masalah yang awalnya mengilhami teori graf. Pertimbangkan "berbeda" Soal: Di bawah ini adalah gambar empat titik yang dihubungkan oleh beberapa garis. Apakah itu mungkin untuk melacak setiap baris sekali dan hanya sekali (tanpa mengangkat pensil, dimulai dan diakhiri dengan titik)?

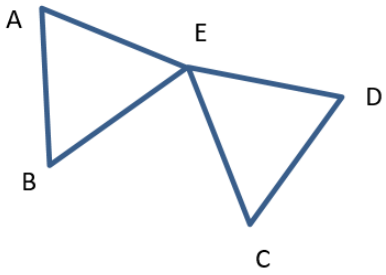


Ada hubungan yang jelas antara kedua masalah ini. Jalan apa saja di titik dan gambar garis sesuai persis dengan jalan di atas jembatan dari Königsberg.

Gambar seperti titik dan gambar garis disebut graph. Graph adalah terdiri dari kumpulan titik-titik yang disebut simpul dan garis yang menghubungkannya titik-titik yang disebut sisi. Ketika dua simpul dihubungkan oleh sebuah sisi, kita katakan mereka berdekatan. Hal yang menyenangkan tentang melihat graph daripada gambar sungai, pulau, dan jembatan adalah bahwa kita sekarang memiliki objek matematika untuk belajar. Kami telah menyaring bagian "penting" dari gambar jembatan untuk tujuan dari memecahkan masalah.

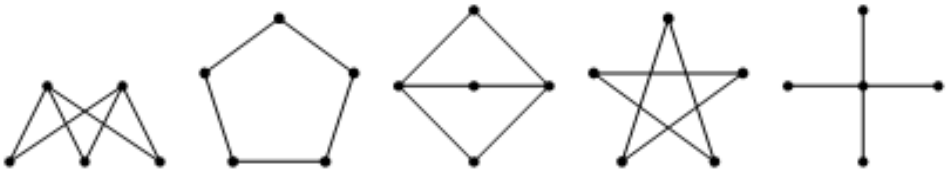
Contoh masalah yang lain,

Misalkan Ali, Adi, Supri, Dan, dan sardi semuanya adalah anggota jejaring sosial di situs web Facebook. Situs ini memungkinkan anggota untuk menjadi "berteman" dengan satu sama lain. Ternyata Ali dan Adi berteman, begitu juga Supri dan sardi. Sardi berteman dengan semua orang. Gambarkan situasi ini dengan grafik. Solusinya. Setiap orang akan diwakili oleh sebuah simpul dan masing-masing persahabatan akan diwakili oleh keunggulan. Artinya, dua simpul akan berdekatan (akan ada tepi di antara mereka) jika dan hanya jika orang yang diwakili oleh simpul tersebut adalah teman

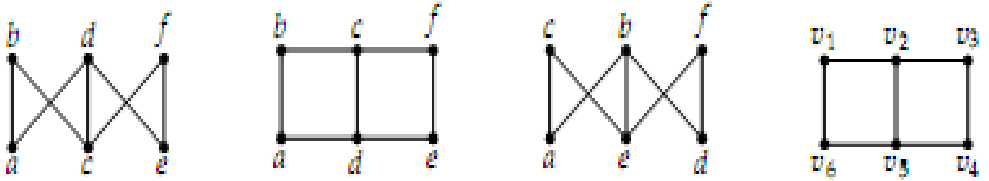


Menyelidiki!

Cobalah aktivitas di atas sebelum melanjutkan! Manakah (jika ada) Graph di bawah ini yang sama?



Graf di atas tidak berlabel. Biasanya kita memikirkan graf memiliki himpunan simpul tertentu. Manakah (jika ada) dari grafik di bawah ini yang sama ?



Sebenarnya, semua gambar yang kita lihat di atas hanyalah gambar Dari graph. Graf benar-benar merupakan objek matematika abstrak terdiri dari dua himpunan V dan E di mana E adalah himpunan dari himpunan bagian 2 elemen dari V. Apakah grafik di bawah ini sama atau berbeda?

Graph 1:

V {a, b, c, d, e},

E {{a, b}, {a, c}, {a, d}, {a, e}, {b, c}, {d, e}}.

Graph 2:

V {v1 , v2 , v3 , v4 , v5},

E {{v1 , v3}, {v1 , v5}, {v2 , v4}, {v2 , v5}, {v3 , v5}, {v4 , v5}}.

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

Definisi Graf.

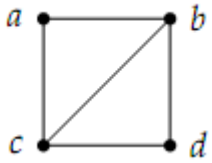
Suatu graf adalah pasangan tak beraturan G = {V, E} terdiri darian set V (disebut simpul) dan himpunan E (disebut tepiatau sisi) dari dua elemen himpunan bagian dari V.

Dari definisi,

$(\{a,b,c,d\},\{\{a,b\},\{a,c\},\{b,c\},\{b,d\},\{c,d\}\}).$

Di sini kita memiliki grafik dengan empat simpul (huruf a; b; c; d) dan lima tepi pasangan sebagaimana berikut,

$\{a,b\},\{a,c\},\{b,c\},\{b,d\},\{c,d\}\}).$



Contoh

$G_1 = (\{a,b,c\},\{\{a,b\},\{b,c\}\});$ $G_2 = (\{a,b,c\},\{\{a,c\},\{c,b\}\}).$

Di sini, himpunan simpul dari setiap graf sama, kedua graf memiliki dua sisi. Di graf pertama, kami memiliki sisi {a, b} dan {b, c}, sementara di grafik kedua kita memiliki tepi {a, c} and {c, b}. Sekarang kita memiliki {b, c} = {c, b} itu bukan masalahnya. Masalahnya adalah {a, b} tidak sama dengan {a, c}. Sehingga himpunan kedua graf tidak sama (sebagai himpunan), grafnya tidak sama (sebagai grafik)

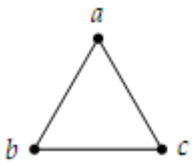
Bahkan jika dua grafik tidak sama, mereka mungkin pada dasarnya sama. grafik pada contoh sebelumnya dapat digambar seperti ini:

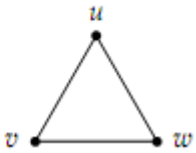
$G_1 = (V_1, E_1)$ where $V_1 = \{a,b,c\}$ and $E_1 = \{\{a,b\},\{a,c\},\{b,c\}\};$

$G_2 = (V_2, E_2)$ where $V_2 = \{u,v,w\}$ and $E_2 = \{\{u,v\},\{u,w\},\{v,w\}\}.$

Contoh lain

Ditunjukkan kan graf sebagai berikut,





1.3 Latihan Soal

1. Jika ditunjukkan A ={1, 2, 3, 4, 5}, B = {3, 4, 5, 6, 7}, dan C = {2, 3, 5}.
   (a) Tunjukkan A ∩ B.
   (b) Tunjukkan Find A ∪ B.
   (c) Tunjukkan A \ B.
   (d) Tunjukkan A ∩ (B ∪ C).
2. Carilah kardinalitas dari setiap himpunan di bawah ini.
   (a) A {3, 4, . . . , 15}.
   (b) B {n ∈ N : 2 < n ≤ 200}.
   (c) C {n ≤ 100 : n ∈ N ∧ ∃m ∈ N(n 2m + 1)}

3. Temukan dua himpunan A dan B dimana |A| = 5, |B| = 6, dan |A ∪ B| = 9. jelaskan |A ∩ B|
4. Tentukan himpunan A dan B dengan |A|= |B| sedemikian rupa sehingga |A ∪ B|= 7 dan |A ∪ B|= 3. Apa itu |A|?
5. Misalkan A = {1, 2, . . . , 10}. Definisikan B2 = {B ⊆ A : |B| = 2}.tentukan |B2 |.
6. Untuk sembarang himpunan A dan B, tentukan AB {ab : a ∈ A ∧ b ∈ B} jika A = {1, 2} and B = {2, 3, 4}, jelaskan |AB| dan |A × B|?
7. Jika ditunjukkan A = {a, b, c, d}. Cari P(A)nya
8. Jika masing-masing 10 orang saling berjabat tangan, berapa kali berjabat tangan? ambil tempat? Apa hubungan pertanyaan ini dengan teori graf?
9. Di antara sekelompok 5 orang, mungkinkah semua orang berteman? dengan tepat 2 orang dalam kelompok? Bagaimana dengan 3 orang? dalam kelompok?
10. Perhatikan dua grafik berikut!

G_1 $V_1 = \{a, b, c, d, e, f, g\}$

$E_1 = \{\{a, b\}, \{a, d\}, \{b, c\}, \{b, d\}, \{b, e\}, \{b, f\}, \{c, g\}, \{d, e\}, \{e, f\}, \{f, g\}\}$.

G_2 $V_2 = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7\}$,

$E_2 = \{\{v_1, v_4\}, \{v_1, v_5\}, \{v_1, v_7\}, \{v_2, v_3\}, \{v_2, v_6\}, \{v_3, v_5\}, \{v_3, v_7\}, \{v_4, v_5\}, \{v_5, v_6\}, \{v_5, v_7\}\}$

Gambarkan masing masing grafnya.

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

Bab 2

Pengantar Teori Bahasa Automata dan Aplikasi Automata

2.1 Kata dan Bahasa

2.1.1 Definisi Bahasa dan Automata

Notasi Alfabet dan Variabel Simbol/String

a. Notasi Alfabet

Alfabet didefinisikan dengan himpunan alphabet dengan menggunakan simbul □. Untuk memudahkan dan penyederhanaan alfabet, hanya berisi bebrrapa simbul a bahkan mungkin satu symbol bahkan nullable. Namun secara umum berlaku untuk alfabet yang lebih besar. Variabel string tertentu sering direpresentasikan sebagai r,s,t, ..., x,y,z. (Huruf kecil di urutan belakang dalam abjad latin). Variabel simbol sering direpresentasikan sebagai a,b,c .. ...yaitu huruf kecil pada urutan terdepan abjad, sebagai contoh,

$$\Sigma = \{x\}$$

$\Sigma = \{x\}$, dapat diartikan bahwa himpunan kata atau string yang terdiri dari barisan factor huruf x saja

$\Sigma = \{a, b, c\}$, himpunan kata atau string yang terdiri dari barisan factor huruf a,b dan c, termasuk reverse dari mereka

$\Sigma = \{a, b, \varepsilon\}$, himpunan kata atau string yang terdiri dari barisan factor huruf a,b dan , ε termasuk reverse dari mereka

b. Variabel Simbol/String

ε

a

abc

abbac

bcbbaaa

babaaaaab

aaabbbaabab

ε

0

1

01

00

100

101

1000

01010101

atau dapat ditulis dengan kata (w), dimana w = Ɛ, w = a, w = abc, w = abbac, w = babaaaaab, dan seterusnya.

c. Operasi String

Length of string atau panjang String, suatu string tersusun atas sejumlan n simbol, n ≥ 0.

Panjang string x dapat ditulis dengan |x|

Misal x = aabaa, maka panjang stringnya adalah |x| = 5

|aab| = 3, panjang stringnya adalah 3, |aaaabbb| =7, panjang stringnya adalah 7

d. Nullstring (String kosong)

Jika |x| = 0 maka disebut String kosong

Biasa dinyatakan dengan ε atau □

|ε| = 0

e. Conkatenasi

Conkatenasi, adalah penyambungan. String x dan y apabila disambungkan akan akan menjadi string xy

Contoh : x = abba, dan y = aa maka xy = (abba)(aa)

Jika x = ab , y = ε maka xy = ab

Konkatenasi Berulang pada String

Menggunakan notasi pangkat

Contoh : x = ab

Maka x4 = (ab)(ab)(ab)(ab)=abababab

(abaa)2 = (abaa)(abaa)=abaaabaa

Konkatenasi pada Himpunan String

Jika bahasa L1 = {aa, bab}, bahasa L2 = {bb,a}

Maka bahasa L1L2 = {aabb, aaa, babbb, baba}

f. Reverse

Merupakan pembalikan string,

Contoh : Jika x = ab, maka Reverse dari xR = ba

Jika x = 0111 maka Reverse dari xR =1110

g. Substring

Substring dari string

Subsequen dari huruf atau karakter yg berurutan

Jika x = acbab, substring dari x adalah ac

Jika x= 10011, substring dari x adalah 10,

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

Jika x = abcd, substring dari x adalah ab

h. Prefix and Suffix

Prefix - substring di awal String x = abbab

ε

a

ab

abb

abba

abbab

Suffix - substring di akhir String x = abbab

abbab

bbab

bab

ab

b

ε

i. Operasi (*)

Himpunan seluruh string yg mungkin dari alphabet □

over alphabet Σ*

Contoh :

Jika alphabet Σ = {x, maka over alphabetnya adalah, sehingga dapat ditulis

Σ* = {himpunan kata yang terdiri dari barisan huruf x saja, termasuk nullstring}, sehingga dapat ditulis sebagai berikut

Σ* ={ Ɛ, x, xx, xxx, xxxx, xxxxx, ……..,……..,…….}

Jika Alpbabet Σ = {a,b}, maka over alphabetnya adalah Σ* = { himpunan kata yang terdiri dari barisan huruf dari factor a a dan b, termasuk reverse dari mereka dan nullstring}

{Ɛ, a,b, aa, ab,ba,bb, aaa, aab, ...,…….,……...}

j. Operasi (+) □+

Himpunan seluruh kata atau string yang mungkin dari alphabet □ kecuali tidak termasuk Ɛ

Σ = {a,b}

Σ* = { Ɛ, a,b, aa, ab,ba,bb, aaa, aab, ...}

Σ+ = Σ * - Ɛ

Σ+ = {a,b,aa, ab,ba,bb, aaa, aab,....}

Kita semua sudah memahami tentang bahasa alami, seperti bahasa Indonesia, Inggris dan Prancis., kebanyakan dari kita merasa sulit untuk mengatakannya dengan tepat apa arti kata "bahasa". Kita dapat mendefinisikan istilah bahasa secara informal sebagai sistem ekspresi, fakta, atau konsep tertentu, termasuk seperangkat simbol, barisan simbul dan aturan untuk manipulasi mereka. Kita membutuhkan definisi yang tepat. Kita mulai dengan himpunan

simbol yang terbatas dan tidak kosong, yang disebut alfabet. Dari simbol kami membangun string yang terbatas urutan simbol dari alfabet {Σ }

Misalnya,

$$\Sigma = \{x\}$$

L1 ={x, xx, xxx, xxxx...}, hal ini dapat ditulis

L1 = { Xn forn = 1 2 3 ... }

Dicontohkan dengan bahasa yang lain

Misalkan Σ = {0,1,2,3,4,5,6,7,8,9}, kita kan mendifinisikan bahasa L 2 = { himpunan string/kata yang terdiri dari factor alphabet} sehingga bahasa itu adalah,

L2 = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, …… }

Jika Σ = 0 (the empty set), maka Σ* ={λ}, hal ini tidak sama dengan,

Jika S = {λ}, maka kemudian S* = {λ}

Kita akan menuliskan sebuah himpunan kata atau string dari alphabet , dengan menunjukkan perbedaan hasil dari notasi + dan Notasi *

Jika terdapat alphabet Σ = {x}, maka dengan demikian

Σ+ = { x, xx. xxx. . . ……}, artinya bahwa himpunan alphabet terdiri dari huruf x, tidak termasuk, huruf null string, sedangkan notasi * adalah

Σ* = { λ, x, xx, xxx, xxxx, …….., ……..}, artinya himpunan huruf yang terdiri dari huruf x, termasuk didalamnya nullstring {λ}

Contoh 1.1

Misalkan Σ= {a, b}. Kemudian Σ*= {λ, a, b, aa, ab, ba, bb, aaa, aab, ...} .

Contoh 1.2.

Jika Σ= {0,1}, maka kemudian Σ * = { λ, 0, 1, 00, 01, 10, 11, 000, 001, …..,……...}

Contoh 1.3.

Σ= {a,b,c}, maka kemudian Σ*={ λ ,a, b, c, aa, ab, ac, ba, bb, bc, ca, cb, cc, aaa,.....,}

Operasi Kleene Star

Kleene star sebagai operasi kata atau bahasa yang tak terbatas, atau himpunan string dari huruf alfabet. Bahasa yang tak terbatas yang dimaksudkan adalah himpunan kata dan panjang katanya tak terhingga.

Penggunaan operator bintang dalam bahasa,

Definisi

Jika S adalah himpunan kata, maka yang dimaksud dengan S* adalah himpunan semua string berhingga yang terbentuk dengan menggabungkan kata-kata dari S, di mana kata apa pun dapat digunakan sebagaimana yang kita inginkan, dan di mana nullstring atau λ juga disertakan.dimana berlaku Untuk set string S apa pun, dimana S* = S**

Contoh 1.3

Jika S = {aa,b}, maka

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

S*= { λ dan semua kata apa saja yang terdiri dari faktor aa dan b }atau dengan kata lain bahwa { λ dan semua string a dan b di mana a muncul dalam rumpun genap }

Contoh 1.4

S = { a, ab }. Maka kemudian S* = { λ ditambah dengan kata apa pun yang terdiri dari faktor a dan ab }, atau dengan pengertian lain yaitu - { λ ditambah semua string a dan b kecuali yang dimulai dengan b dan yang mengandung b ganda. Bahasa itu adalah,

S* = { λ, a, aa, ab, aaa, aab, aaaa, aaab, aaba, abaa, abab,aaaaa, aaaab, aaaba, aabaa, aabab, abaaa, abaab, ababa,.....,……..,…….. }

Penggunaan notasi notasi +

Jika Σ={x}, maka Σ+ = { x, xx, xxx. . . …,…..,….. }

Contoh 1.5.

S = {w1, w2, w3 }, maka kemudian

S+ = {w1, w2, w3, w1 w1, w1 w2, w1 w3, w2 w3, w3 w2, w3 w3, w1 w1 w1, w1 w2 w3, ……., ……..}

Jika w1= *aa,* w2 = *bbb,* w3 = λ , maka S+ = { *aa, bbb,* λ, *aaaa, aabbb*…….}

Panjang dari string

Definisi "panjang string" sebagai jumlah huruf dalam string fungsi ini menggunakan kata "panjang". Misalnya, jika a = xxxx , maka panjang a, adalah 4, hal ini dapat di tulis, panjang kata (a) = 4

Contoh1.2.

Jika c = 428, maka di dalam bahasa L terdapat panjang kata (c)=3, demikian pula jika panjang kata (xxxx) =4, untuk panjang kata {λ} = 0, besaran dari |λ| = 0, w = wλ = w

Penggabungan string/kata

Jika alfabet Σ = {a, b}, maka abab dan aaabbba adalah string-string atau kata dalam suatu bahasa, misalnya, w = abaaa untuk menunjukkan bahwa string bernama w memiliki nilai spesifik abaaa. Penggabungan dua string w dan v adalah string yang diperoleh dengan menambahkan simbol v ke ujung kanan w, yaitu, jika w = a_1a_2 • • • • an dan v = b_1b_2 • • • b_m, maka rangkaian w dan v, dilambangkan dengan wv, adalah wv = a_1a_2 • • • an b_1b_2 • • • b_m.

2.2 Hirarki Chomsky

Tata bagasa (Grammer), Grammar dapat didefinisikan sebagai kumpulan dari kumpulan variabel, atau simbul Non terminal ,simbol terminal, simbol awal, kemudia terdapat aturan -aturan produksi, terdapat empat Type tata bahasa atau bahasa menurut Chomsky, yaitu (Brookshea 1989)

1) Tipe 0 – Unrestricted Grammar
2) Tipe 1 – Context Sensitive Grammar
3) Tipe 2 – Context Free Grammar
4) Tipe 3 – Regular Grammar
5) Tipe 0 – Unrestricted Grammar

Aturan produksinya tidak memiliki batasan dengan aturan $\alpha \rightarrow \beta$.

sebagai contoh aturan-aturan produksinya sebagai berikut;

S → ABaA

Ac → AcB

CA → DD

bD → Da

SBa → ab

A → S

Tipe 1 – Context Sensitive Grammar

Aturan produksi-produksinya |a|≤|b| (jumlah simbol di ruas kiri harus lebih kecil atau sama dengan jumlah simbol di ruas kanan)

Aturan S → Ɛ dibolehkan jika S tidak muncul pada ruas kanan setiap aturan

sebagai contoh aturan-aturan produksinya sebagai berikut;

S → AC

AD → abcde

A → a

aB → Db

aD → bb| bbCC

Bc → Cabb (Terpenuhi)

Tipe 2 – Context Free Grammar

Aturan produksi ini memiliki 1 variabel non terminal sebelah kiri kemudian diikuti dengan sebelah kanan

contoh:

- S → X a
- X → a
- X → aX
- X → abc
- X → Є
- S → AB

Tipe 3 – Regular Grammar

Aturan produksi ini memiliki 1 variabel non terminal sebelah kiri kemudian diikuti dengan sebelah kanan , beberapa terminal saja, terminal Non Terminal, atau terminal beberapa Non terminal bahkan Nollstring (Є)

Contoh:

S →abb | aY

Y → b

S → abB

X → Ɛ

Latihan soal-soal

1) Tunjukkan bahasa S*, di mana S = {a, b}. Berapa banyak kata yang dimiliki bahasa ini dengan panjang 2? panjang 3? dari panjang n?
2) Tunjukkan bahasa S*, di mana S = {aa, b}. Berapa banyak kata yang dimiliki bahasa ini dengan panjang 4? panjang 5? dari panjang 6? Apa yang bisa dikatakan secara umum?
3) Tunjukkan bahasa S*, di mana S = {ab, ba}. Tulis semua kata dalam S* yang memiliki tujuh huruf atau kurang. Bisakah kata apa saja dalam bahasa ini? mengandung substring aaa atau bbb?
4) Tunjukkan bahasa S*, di mana S = {a, ab ,ba}. Apakah string (abbba) sebuah kata dalam bahasa ini? Tulis semua kata dalam bahasa ini dengan tujuh atau kurang huruf. Apa cara lain untuk menggambarkan kata-kata dalam bahasa ini? Hati-hati, ini bukan hanya bahasa semua kata tanpa bbb.

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

2.3 Aplikasi Automata

Beberapa game permainan dadu yang dimainkan anak-anak. Dimana diatur di atas papan dadu permainan, Dadu dilempar (atau atau dadu diputar), dan sebuah angka dihasilkan secara acak. Tergantung pada jumlahnya, potongan-potongan di papan dadu harus diatur dengan cara yang sepenuhnya ditentukan oleh aturan. Kita tidak boleh memiliki pilihan tentang mengubah papan dadu. Semuanya ditentukan oleh dadu. Biasanya giliran anak lain yang melempar dadu,Kita bisa mengalahkan lawan. Apakah kita memenangkan permainan atau tidak adalah tergantung sepenuhnya pada urutan angka apa yang dihasilkan oleh dadu, bukan pada siapa yang yanglebih dulu menggerakkan anak dadu. (Rich 2008)

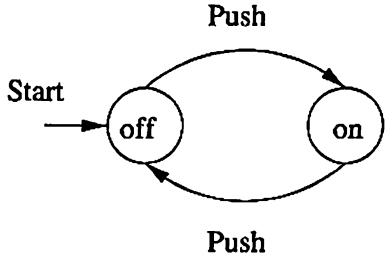
Permainan berubah dari satu keadaan ke keadaan lain dengan cara yang ditentukan dengan memasukkan angka tertentu. Untuk setiap nomor yang mungkin ada satu dan hanya satu keadaan yang dihasilkan. Kita harus mengizinkan kemungkinan bahwa setelah nomor dimasukkan permainan masih dalam keadaan yang sama seperti sebelumnya.(Misalnya, jika seorang pemain yang berada di "posisi kalah" perlu melakukan roll ganda untuk keluar, gulungan lainnya meninggalkan papan dadu dalam keadaan yang sama.) Setelah beberapa jumlah gulungan, papan dadu tiba pada keadaan yang berarti kemenangan untuk salah satu para pemain dan permainan selesai. Kami menyebutnya keadaan akhir atau state akhir/state penerimaan. (Hopcroft et al. 2001)

Contoh yang lain, Seorang anak memiliki komputer sederhana (input device, processing unit, memory, output device)dan ingin menghitung jumlah 3 ditambah 4. Anak itu menulis sebuah program,yang merupakan urutan instruksi yang dimasukkan ke dalam mesin satu per satu waktu. Setiap instruksi segera Dieksekusi setelah dibaca, dan kemudian instruksi berikutnya dibaca. Jika semuanya berjalan dengan baik, mesin mengeluarkan angka 7 dan mengakhiri eksekusi. (Rich 2008)

Contoh yang lain yang sederhana, finite Automata juga sebuah model yang dapat digunakan untuk membantu banyak bidang diantaranya konsep perangkat keras dan perangkat lunak. Kita akan mempelajari contoh-contoh bagaimana konsep-konsep itu digunakan. Untuk saat ini,:

a. Perangkat lunak untuk merancang dan memeriksa perilaku sirkuit digital.
b. Analisis leksikal dari kompiler, yaitu komponen kompiler yang memecah teks input menjadi unit logis, seperti pengidentifikasi,kata kunci, dan tanda baca.
c. Perangkat lunak untuk memindai teks dalam jumlah besar, seperti kumpulan halaman Web, untuk menemukan kemunculan kata, frasa, atau pola lainnya.
d. Perangkat lunak untuk sistem verifikasi dari semua jenis yang memiliki jumlah terbatas state yang berbeda, seperti protokol komunikasi atau protokol untuk keamanan Pertukaran informasi.

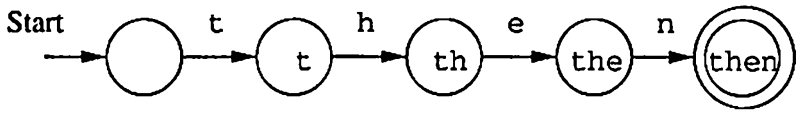
Sementara kita akan segera menemukan definisi yang tepat dari berbagai jenis automata, mari kita mulai perkenalan sketsa tentang mesin automata yang sangat sederhana, seperti yang disebutkan di atas, yang dapat dilihat setiap saat dalam salah satu dari jumlah finite state. Misalnya, kita bisa mengimplementasikannya di perangkat keras sebagai sirkuit, atau sebagai bentuk program sederhana yang dapat membuat keputusan hanya dengan melihat menggunakan posisi dalam kode itu sendiri untuk membuat keputusan. contoh sederhana adalah on/off, pengalihan sircuit sebagaimana gambar 1, dan memungkinkan kita menekan tombol yang berbeda, tergantung pada keadaan saklar. Artinya, jika sakelar dalam keadaan off, maka tekan tombol mengubahnya ke state aktif artinya posisinya hidup, dan jika sakelar dalam state aktif, maka menekan tombol yang sama mengubahnya ke keadaan mati. (Peter 2008)



Gambar 2.1: Finite automata model sakelar hidup/mati

Model finite automata untuk sakelar ditunjukkan pada Gambar. 1 Untuk semua finite automata, keadaan diwakili oleh lingkaran; dalam contoh ini, kita memiliki state dan mematikan. Busur antar dari state ke stste diberi label "input PUSh", yang mewakili pengaruh eksternal pada sistem. Di sini, kedua busur diberi label oleh input Push, yang mewakili pengguna yang menekan tombol. kita bisa menganggap state pada Gambar sebagai menerima, karena dalam state itu, perangkat dikendalikan oleh saklar.(Hopcroft et al. 2001)

Pada contoh automata gambar 2, mesin automata membaca string then yaitu bagian dari penganalisis leksikal. Tugas mesin finite automata ini adalah mengenali kata. Dengan demikian membutuhkan lima state, yang masing-masing berbeda posisi yang telah dicapai berupa input t, h, e dan n sampai meraih state akhir, yaitu then, gambar ini menunjukkan bagaimana mesin automata ini mengenal kata then yaitu berangkat dari initial state dan meraih pada state akhir atau state penerimaan, yaitu state Then sebahgai stete penerimaan, ditandai denga lingkaran doble bulat.(Sumarno and Prasetiyo 2019)



Gambar 2.2: Pengenalan pemodelan finite automata

Gambar 2, lima stete diberi nama stete awal. Diberikan input huruf. Kita dapat membayangkan bahwa penganalisis leksikal memeriksanya karakter program yang dikompilasi pada suatu waktu, dan karakter berikutnya yang akan diperiksa adalah input state automata. State awal sesuai dengan string kosong, dan setiap state dapat input huruf menuju state berikutnya hingga meraih state then tersebut.

Di bagian lain, kita akan mempelajari contoh yang diperluas dari masalah dunia nyata yang solusinya menggunakan finite automata. Kami menyelidiki protokol yang menggambarkan "uang elektronik" yang dapat digunakan pelanggan untuk membayar untuk barang lewat teknologi internet, penjual dapat menerima jaminan uang. Penjual harus tahu bahwa uang bentuk file tersebut tidak dipalsukan, juga tidak disalin dan dikirim ke penjual, sementara pelanggan menyimpan salinannya dari bentuk file yang sama untuk dibelanjakan lagi. File yang tidak dapat dipalsukan adalah sesuatu yang harus dijamin oleh bank. Artinya, ada tiga entitas yang terlibat, bank, harus mengeluarkan dan mengenkripsi file "uang", sehingga pemalsuan tidak terjadi. Namun, bank memiliki pekerjaan penting, bank harus menyimpan database semua uang yang valid yang telah dikeluarkan, sehingga dapat memverifikasi ke toko bahwa uang bentuk file telah diterima mewakili uang nyata dan dapat dikreditkan ke akun toko. Namun, untuk menggunakan uang elektronik, protokol perlu dirancang untuk memungkinkan manipulasi uang dalam berbagai cara yang diinginkan oleh pengguna. Karena sistem moneter selalu mengundang penipuan, kita harus memverifikasi kebijakan apa pun kita mengadopsi tentang bagaimana uang digunakan. Artinya, kita perlu membuktikan satu satunya hal-hal yang bisa terjadi adalah hal-hal yang kita inginkan terjadi hal-hal yang memungkinkan tidak terjadi, pengguna yang tidak jujur untuk mencuri dari orang lain atau untuk menghasilkan uang. Dalam keseimbangan bagian ini, akan memperkenalkan contoh yang sangat sederhana dari protokol uang elektronik yang buruk, dengan finite automata, dan ditunjukkan bagaimana konstruksi pada automata dapat digunakan untuk memverifikasi protokol (atau, dalam hal ini, untuk menemukan bahwa protokol memiliki bug.

Ada tiga peserta: pelanggan, toko, dan bank. diasumsi bahwa hanya ada satu file "uang" yang ada. Pelanggan dapat memutuskan untuk mentransfer file uang ini ke toko, yang kemudian akan menebus file dari bank (yaitu, meminta bank untuk mengeluarkan file uang baru milik toko dari pelanggan) dan toko mengirimkan barang ke pelanggan. Pada suatu kejadian pelanggan dapat memiliki opsi untuk membatalkan file uang. Artinya, pelanggan mungkin menghubungi pihak bank untuk menempatkan uang kembali di rekeningnya, membuat uang tidak lagi dibelanjakan ke toko.

Interaksi di antara tiga peserta dengan demikian terbatas menjadi lima acara:

a. Pelanggan dapat memutuskan untuk membayar. Yaitu pelanggan mengirim file uang ke toko,
b. Pelanggan dapat memutuskan untuk membatalkan. File uang dikirim ke bank dengan pesan bahwa nilai file uang akan ditambahkan ke pelanggan dengan akun bank.
c. Toko dapat mengirimkan barang kepada pelanggan.

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

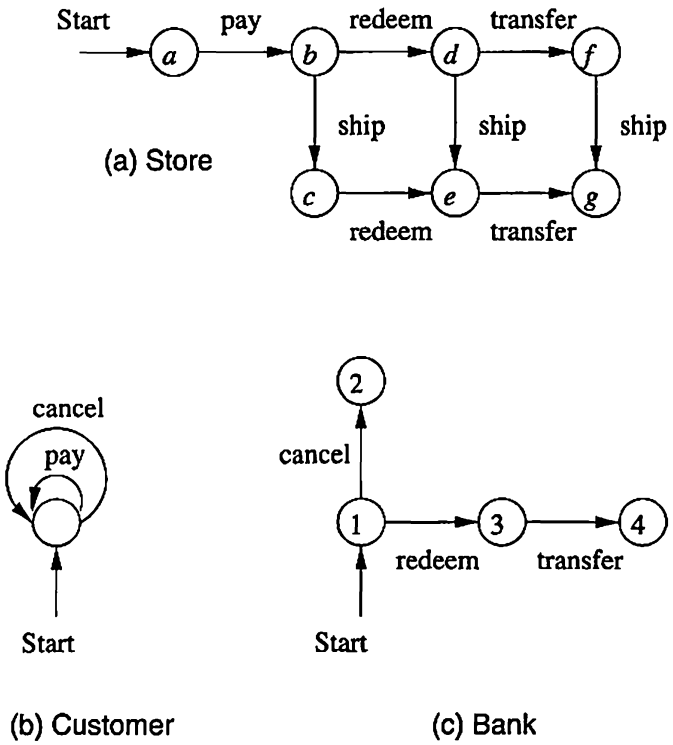
d. Toko dapat menebus file uang. Artinya, file uang itu dikirim ke bank dengan permintaan agar nilainya diberikan ke toko.
e. Bank dapat mentransfer uang dengan membuat yang baru, terenkripsi yang sesuai file uang dan mengirimkannya ke toko.

Protokol Ketiga peserta harus merancang aturan dengan hati-hati, atau hal-hal yang salah mungkin terjadi. Dalam contoh, diasumsikan bahwa pelanggan bisa tidak bertanggung jawab, pelanggan mungkin mencoba menyalin file uang, menggunakannya untuk membayar beberapa kali pembayaran pembelian pada toko, atau membayar dan membatalkan pembayaran dengan uang file uang, sehingga mendapatkan barang secara gratis. Bank harus berperilaku bertanggung jawab, Secara khusus, bank harus memastikan bahwa dua toko tidak dapat menukarkan file uang yang sama, Toko harus hati-hati, toko tidak boleh mengirimkan barang sampai ia yakin telah menerima uang dari barang dagangan tersebut.

Protokol jenis ini dapat direpresentasikan sebagai Finite automata. Setiap state mewakili situasi yang mungkin dialami oleh salah satu peserta/entitas. Artinya, state "memory" bahwa peristiwa penting tertentu telah terjadi dan yang lain belum terjadi. Transisi antar state terjadi ketika salah satu dari lima peristiwa dijelaskan di atas terjadi. Kami akan menganggap peristiwa ini sebagai "eksternal" untuk automata mewakili ketiga peserta/entitas, meskipun masing-masing peserta adalah bertanggung jawab untuk memulai satu atau lebih peristiwa perubahan.

Gambar 2.1 mewakili tiga peserta dengan automata. Dalam diagram itu, hanya menampilkan peristiwa yang memengaruhi peserta. Misalnya, tindakan membayar hanya mempengaruhi pelanggan dan toko.

Bank tidak tahu bahwa uang itu telah dikirim oleh pelanggan ke toko atau tidak. Mari kita periksa dulu outomata (c) untuk bank. Kita mulai dari state awal adalah State 1; menjelaskan situasi di mana bank telah mengeluarkan file nilai uang di pertanyaan, tetapi belum diminta untuk membayar atau membatalkannya.

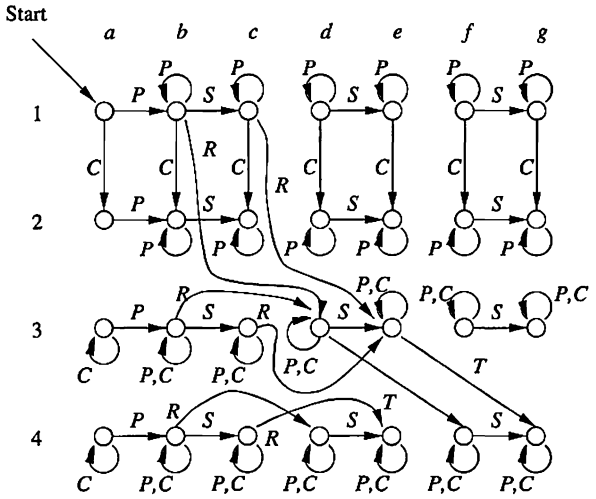


Gambar 2.3: Konsep Automata aksi bisnis masing-masing Peserta b

Jika sebuah permintaan batal dikirim ke bank oleh pelanggan, kemudian bank mengembalikan uang ke rekening pelanggan dan memasuki state 2. State terakhir mewakili situasi di mana uang telah dibatalkan. Bank, yang bertanggung jawab, tidak akan meninggalkan state 2 setelah dimasukkan, karena bank tidak boleh mengizinkan hal yang sama uang untuk dibatalkan lagi atau dibelanjakan oleh pelanggan. Atau, ketika di berada di state1 bank dapat menerima permintaan penebusan dari toko. Jika demikian, ia menuju ke state 3, dan segera mengirim toko

pesan transfer, dengan file uang baru yang sekarang menjadi milik toko. Setelah mengirim pesan transfer, bank pergi ke State 4. Dalam keadaan itu, ia tidak akan menerima permintaan pembatalan atau tidak akan melakukan tindakan lain apa pun terkait hal nilai uang. Sekarang, mari kita perhatikan Gambar 2.1 (a), Mesin Automaton yang mewakili aksi dari toko. Sementara bank selalu melakukan hal yang benar, sistem toko memiliki beberapa cacat. Bayangkan bahwa pengiriman dan operasi keuangan dilakukan oleh proses yang terpisah, sehingga ada peluang untuk tindakan yang dilakukan baik sebelum, sesudah, atau selama penukaran uang elektronik. Itu policy memungkinkan toko untuk masuk ke situasi di mana ia telah mengirimkan barang dan kemudian menemukan uang itu palsu.

Toko dimulai di state a. Ketika pelanggan memesan barang dengan melakukan tindakan pembayaran, toko memasuki state b. Dalam keadaan ini, toko memulai proses pengiriman dan penukaran. Jika barang dikirim, kemudian toko memasuki state c, di mana ia masih harus mengambil uang dari bank dan menerima transfer file uang dari bank. atau, toko dapat mengirimkan pesan tebusan terlebih dahulu, dengan memasukkan state d. Dari state d, toko mungkin mengirim berikutnya, memasuki state c, atau mungkin berikutnya menerima transfer uang dari bank, memasuki state d kami berharap bahwa toko pada akhirnya akan mengirim, menempatkan toko dalam status g, di mana transaksi selesai dan tidak ada lagi yang akan terjadi. Di state e, tokonya adalah menunggu transferan dari bank. Sayangnya, barangnya sudah dikirim, dan jika transfer tidak pernah terjadi, toko kurang beruntung.



Gambar 2.4:

Terakhir, kita amati outomata untuk pelanggan, Gambar 2.1(b). Outomata ini hanya memiliki satu state, yang mencerminkan fakta bahwa pelanggan "dapat melakukan apa saja, Pelanggan dapat melakukan pembayaran dan membatalkan beberapa kali dalam suatu waktu. Pada gambar selanjutnya yaitu gambar 3.2, sebuah sistem yang menghubungkan 3 entitas antara toko, bank dan pelanggan. Kami memiliki model tentang bagaimana ketiga entitas berkomunikasi, sebagaimana disebutkan, karena pelanggan tidak memiliki resiko perilaku, mesin aotomata itu hanya memiliki satu state, dan urutan kejadian apa pun memungkinkannya tetap berada di state itu; Namun, baik state toko dan state bank berperilaku yang kompleks, Misalnya, keadaan C, d)dari outomatatomat produk mewakili situasi di mana bank dalam keadaan 3, dan toko dalam keadaan d. Karena bank memiliki empat state bagian dan toko memiliki tujuh, otomat produk memiliki 4 x 7 = 28 state. Kami menunjukkan otomat produk pada Gambar. 2.3. Untuk kejelasan, kami telah mengatur 28 state bagian dalam array. Baris sesuai dengan keadaan bank dan kolom ke keadaan toko. Untuk menghemat ruang, kami juga telah menyingkat label pada busur, dengan P, 5, C, /?, dan T berdiri untuk membayar, mengirim, membatalkan, menebus, dan mentransfer, masing-masing. Untuk membangun busur otomat produk, kita perlu menjalankan bank dan simpan automata "secara paralel." Masing-masing dari dua komponen produk mesin automatasecara mandiri membuat transisi pada berbagai input. Namun, itu penting untuk diperhatikan bahwa jika tindakan input diterima, dan salah satu dari keduanya automata tidak memiliki status untuk digunakan pada input itu, maka produk automaton "mati"; tidak memiliki state bagian untuk dituju.Untuk membuat aturan transisi keadaan ini tepat, misalkan produk mesin automatadalam keadaan (i^x). Keadaan itu sesuai dengan situasi di mana bank dalam keadaan i dan toko dalam keadaan x. Biarkan Z menjadi salah satu tindakan input. Kita lihat outomata untuk bank, dan lihat apakah ada

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

transisi keluar dari state i dengan label Z. Misalkan ada, dan itu mengarah ke state j (yang mungkin menjadi sama seperti i jika bank loop pada input Z). Kemudian, kami melihat toko dan lihat apakah ada busur berlabel Z yang mengarah ke beberapa keadaan y. Jika keduanya j dan y ada, maka otomat produk memiliki busur dari keadaan (iyx) ke keadaan 0",y), berlabel Z, Jika salah satu dari keadaan j atau y tidak ada (karena bank atau toko tidak memiliki busur dari i atau X, masing-masing, untuk input Z), maka tidak ada busur keluar dari (i,x) berlabel Z.

Sekarang kita dapat melihat bagaimana busur pada Gambar 2.3 dipilih. Misalnya, pada masukan pembayaran, toko beralih dari status a ke 6, tetapi tetap bertahan jika berada di negara lain negara selain a. Bank tetap dalam keadaan apa pun ketika inputnya adalah par/j karena tindakan tersebut tidak relevan dengan bank. Pengamatan ini menjelaskan empat busur berlabel P di ujung kiri dari empat baris pada Gambar 2.3, dan loop berlabel P di negara bagian lain.

Untuk contoh lain tentang bagaimana busur dipilih, pertimbangkan input redeem.

Jika bank menerima pesan penebusan ketika dalam keadaan 1, itu menuju ke keadaan 3. Jika dalam menyatakan 3 atau 4, itu tetap di sana, sementara di negara bagian 2 otomat bank mati; yaitu, memiliki tempat untuk pergi. Toko, di sisi lain, dapat membuat transisi dari keadaan 6 ke d atau dari c ke e saat input redeem diterima. Pada Gambar 2.3, kita melihat enam busur berlabel tebus^ sesuai dengan enam kombinasi dari tiga negara bank dan dua status penyimpanan yang memiliki busur yang terikat ke luar berlabel R, Misalnya, dalam keadaan A,6), busur berlabel R mengambil otomat ke keadaan C,d), karena tebus mengambil bank dari negara 1 ke 3 dan toko dari b ke d. Sebagai contoh lain, ada busur berlabel R dari D, c) ke D, e), karena tebusan mengambil bank dari state 4 kembali ke state 4, sementara itu mengambil store dari state c ke state c.

Menggunakan Product Automaton untuk Memvalidasi Protokol

Pada gambar 3.2 menunjukkan kepada kita beberapa hal yang menarik. Misalnya, dari 28 state, hanya sepuluh di antaranya dapat diraih dari state awal, yaitu (1,a) dari kombinasi state awal bank dan automata toko. Perhatikan bahwa menyatakan seperti (2,e) dan (4,d) tidak dapat diakses^ yaitu. tidak ada jalan menuju mereka dari keadaan awal. Status yang tidak dapat diakses tidak perlu dimasukkan ke dalam robot, dan kami melakukannya dalam contoh ini hanya untuk menjadi sistematis. Namun, tujuan sebenarnya dari menganalisis protokol seperti ini menggunakan automata adalah untuk bertanya dan menjawab pertanyaan yang artinya “bisakah berikut tyi)e kesalahan terjadi?" Dalam contoh yang ada, kita mungkin bertanya apakah itu mungkin bahwa toko dapat mengirimkan barang dan tidak pernah dibayar. Artinya, dapatkah produk itu? robot masuk ke keadaan di mana toko telah dikirim (yaitu, keadaan adalah di kolom c, e, atau g), namun tidak ada transisi pada input T w^ yang pernah dibuat atau akan dibuat? Misalnya, di negara C,e), barang telah dikirim, tetapi akan ada akhirnya menjadi transisi pada input T ke state D,5). Dalam hal apa bank itu lakukan, setelah mencapai status 3, ia telah menerima permintaan tebusan dan diproses itu. Artinya harus dalam keadaan 1 sebelum menerima tebusan dan oleh karena itu pesan pembatalan belum diterima dan akan diabaikan jika diterima di masa depan. Dengan demikian, bank pada akhirnya akan melakukan transfer uang ke toko. Namun, keadaan B,c) adalah masalah. Negara dapat diakses, tetapi satu-satunya busur out mengarah kembali ke keadaan itu. Keadaan ini sesuai dengan situasi dimana bank menerima pesan pembatalan sebelum pesan tebusan. Namun, toko menerima pembayaran moi> bijak: yaitu, pelanggan sedang mendua dan memiliki keduanya menghabiskan dan membatalkan uang yang sama.

Toko dengan mudahnya dan cerobohnya mengirim sebelum mencoba untuk menebus uang, dan ketika toko melakukan tindakan penebusan, bank bahkan tidak akan mengakui pesan tersebut, karena dalam keadaan 2, di mana itu telah membatalkan uang dan tidak akan memproses permintaan tebusan.

Automata untuk Vending Machine

Gambar 2.5: Vending Machine di Jepang

Vending Machine di jepang ini merupakan penerapan bagaimana mesin menerima kata kemudian memilih kata yang tepat sesuai aturan yang berlaku pada mesin, mesin ini menjual barang atau kebutuhan manusia secara otomatis. Sistem penjualan dengan Vending Machine tidak membutuhkan operator, pembeli dapat memilih sendiri barang yang diinginkan. Seiring dengan itu banyak penelitian dilakukan oleh para peneliti, diantaranya Tatas Hari Wicaksono, Faisol Dwiki Amrizal, Hani Atun Mumtahana (2019), Pemodelan Vending Machine dengan Metode FSA (Finite State Automata).

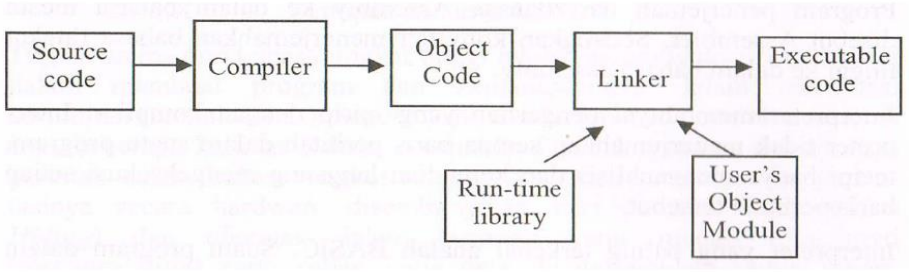
2.3.1 Bahasa Pemrograman dan Kompiler

Dalam Bahasa pemrograman sering dijelaskan dengan Tata bahasa yang khusus yaitu tata bebas kontek. Penyesuaian atau pencocokan tatabahasa bebas kontek yang tatabahasa regulat yang dibangun di banyak lingkungan pemrograman modern. Diagram transisi finite state yaitu sadari suatu state menuju state berikutnya memungkinkan digunakan pada pemrograman visual.

Mendefinisikan Bahasa Pemrograman dengan Sintaksisnya, Sebagian besar bahasa pemrograman adalah bahasa bebas kontek. Ada beberapa bahasa atau tipe bahasa menurut chomsy, yang tidak dapat dijelaskan dalam kerangka bahasa bebas kontek. Tetapi tata bahasa bebas konteks (CFG) memberikan dasar untuk mendefinisikan sebagian besar sintaksisnya dari bahasa pemrograman.

Teknik Kompilasi

Teknik kompilasi merupakan teknologi yang dapat digunakan dalam melakukan pembacaan suatu program yang telah ditulis dalam bahasa sumber, kemudian diterjemahkan ke dalam suatu bahasa pemrograman yang lain yang disebut bahasa sasaran. Sedangkan Compiler merupakan suatu bahasa program yang dapat membaca suatu source language atau Bahasa pemrograman dan kemudian diterjemahkan ke dalam Bahasa pemrograman lain, sebagai contoh Kompiler merupakan program yang menerjemahkan bahasa seperti Pascal, C, PL/I, FORTRAN atau COBOL ke dalam bahasa mesin, kompiler akan menyediakan subroutine khusus dan subroutine tsb hanya akan digunakan pada saat program hasil kompilasi dijalankan. Kumpulan subroutine tsb dinamakan run-time library.



Gambar 2.6: Skematis Proses Kompilasi

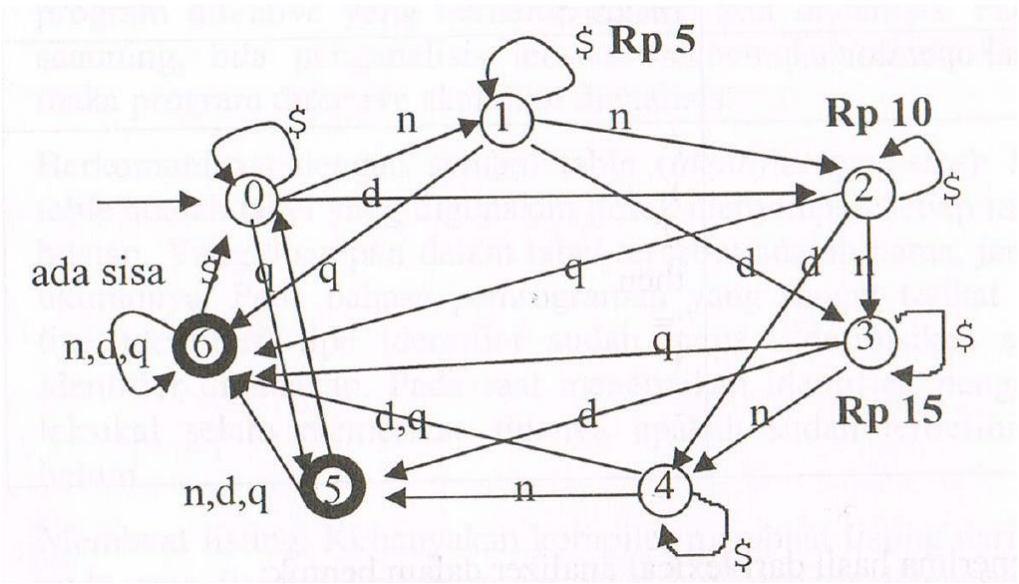
Tahapan Kompilasi

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

- Analisis Leksikal
- Analisis Sintaktik/Semantik
- Intermediate Code Generation
- Optimization
- Object Code Generation

Pada tahapan Leksikal ini tugas utama penganalisis leksikal adalah utuk memecah tiap baris source menjadi token-token, kemudian melakukan membuang komentar, menyeragamkan huruf capital menjadi huruf kecil atau sebaliknya, membuang white space, menginterpretasikan compiler direktif interaksi atau komunikasi dengan table simbul dan membuang listing.

Pada Analisis leksikal ini mudah diimplementasikan pada Finite State Machine automata, dalam hal ini mempelajari sehimpunan state beserta dengan aturan-aturan perpindahan atau transisi dari satu state menuju ke state yang lainnya. Sehimpunan state tersebut menyatakan satu proses dan aturan aturannya menyatakan kemungkinan yang terjadi dalam menyelesaikan proses tersebut, gambaran analisi leksikal ini dapat digambarkan dengan mesin automata sebagaimana menggambarkan mesin penjual permen yang memuat aturan-aturan sebagai berikut: harga permen Rp 25. Mesin tersebut dapat dimasuki 3 jenis koin, Rp 5 (n), Rp 10 (d), Rp 25 (q). $ = tombol untuk mengeluarkan permen. Kemungkinan2 yang terjadi dalam proses tersebut digambarkan dalam state diagram berikut ini:



Gambar 2.7: Konsep Finite Automata untuk mesin penjualan Permen

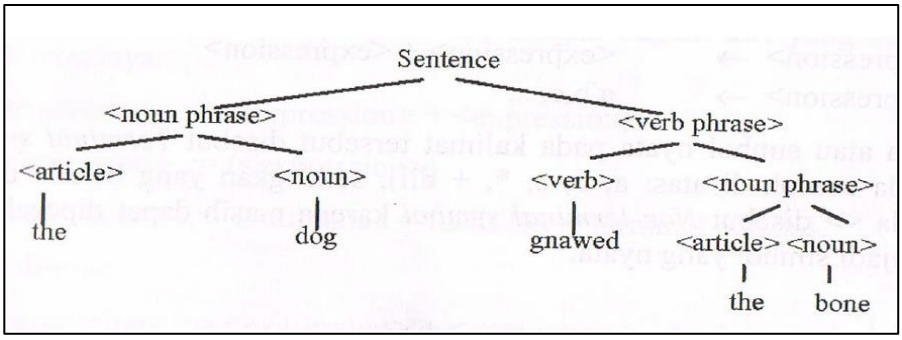
Tabel 2.1: Transisi State penjualan permen

Current state	Input			Tekan tobol ($)
	Rp. 5 (n)	Rp. 10 (d)	Rp. 25 (q)	
0	1	2	5	0
1	2	3	6	1
2	3	4	6	2
3	4	5	6	3
4	5	6	6	0
5	6	6	6	0*
6	6	6	6	0*

Pada tahapan analisis sintek

Sintak adalah susunan kalimat dan aturan-aturan dalam membentuk kalimat disebut grammar. Penganalisis sintak dalam bidang teknik kompilasi sering disebut dengan Parser. • Untuk menganalisis kalimat biasanya digunakan bantuan parse-tree. Atau pohon sintek.

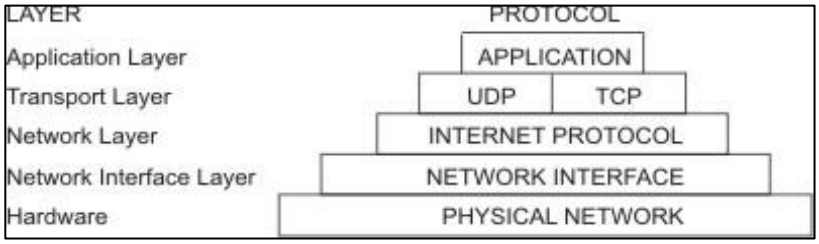
Sebagai contoh, kita peerhatikan kalimat berikut ini, "The dog gnawed the bone". Kalimat tersebut disusun dalam bentuk pohon sintek sebagaimana berikut ini:



Penggunaan Pada Protokol Jaringan Komputer

Dalam teori bahasa dan automata yang dibahas ini dapat digunakan dalam menggambarkan struktur jaringan, sekarang ini akan kita perkenalkan beberapa di antaranya untuk mendifinisikan protocol komunikasi jaringan, monitoring dan memelihara jaringan, mengeksploitasi sumber daya jaringan pemecahan masalah-Web Semantic, dan keamanan jaringan, hacker dan virus.

Protokol adalah seperangkat aturan untuk format pesan dan prosedur yang memungkinkan mesin dan program aplikasi untuk bertukar informasi. Aturan aturan ini harus diikuti oleh setiap mesin yang terlibat dalam komunikasi agar host penerima dapat memahami pesan. Rangkaian protokol TCP/IP dapat dipahami dalam hal lapisan (atau level, Gambar ini menggambarkan lapisan protokol TCP/IP. Dari atas mereka adalah, Application Layer, Transport Layer, Network Layer, Network Interface Layer, dan Network Layer.



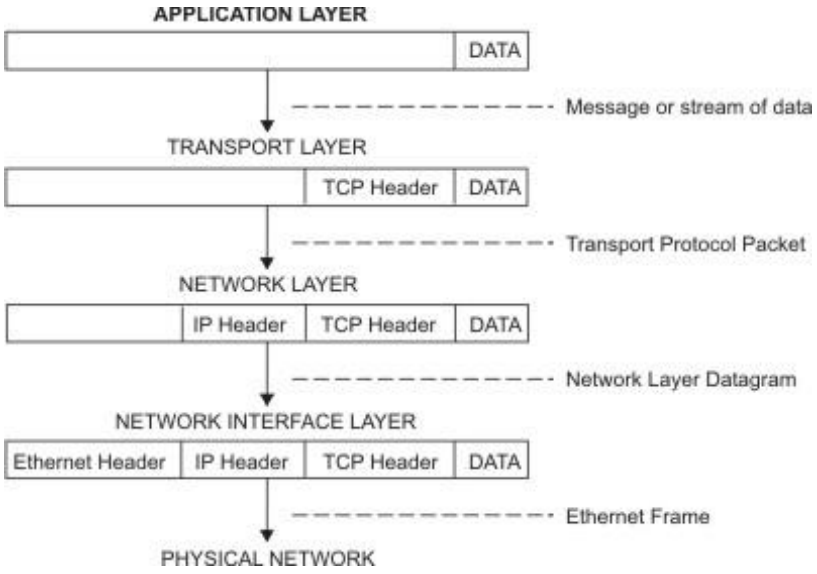
Gambar 2.8: Lapisan protokol TCP/IP

TCP/IP dengan hati-hati mendefinisikan bagaimana informasi berpindah dari pengirim ke penerima. Pertama, program aplikasi mengirim pesan atau aliran data ke salah satu Internet Transport Layer Protocol, baik User Datagram Protocol (UDP) atau Transmission Control Protocol (TCP). Protokol-protokol ini menerima data dari aplikasi, membaginya menjadi bagian-bagian yang lebih kecil yang disebut paket, menambahkan alamat tujuan, dan kemudian meneruskan paket ke lapisan protokol berikutnya, lapisan Jaringan Internet.

Lapisan Jaringan Internet membungkus paket dalam datagram Protokol Internet (IP), memasukkan header dan trailer datagram, memutuskan ke mana harus mengirim datagram (baik langsung ke tujuan atau ke gateway), dan meneruskan datagram ke Lapisan Antarmuka Jaringan.

Lapisan Antarmuka Jaringan menerima datagram IP dan mengirimkannya sebagai bingkai melalui perangkat keras jaringan tertentu, seperti jaringan Ethernet atau Token-Ring.

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress



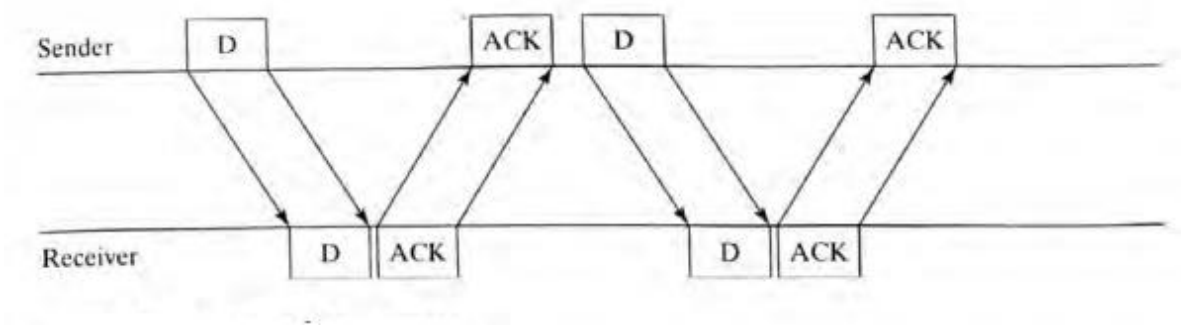
Gambar 2.9: Pergerakan Informasi Dari Pengirim a

Jenis protokol komunikasi jaringan computer TCP/IP ataupun OSI yang sangat bermanfaat hal demikian dapat dimodelkan dengan Mesin finite automata yang bergerak membaca membaca even menuju state yang lainnya dari mesin yang tidak pernah berhenti sesuai dengan even apa yang terjadi. Setiap proses (mesin) menerima input even berupa pengiriman data dan menerimaan data. Jadi, lebih tepatnya, model yang akan kita bangun dengan konsep teori bahasa dan automata automata.

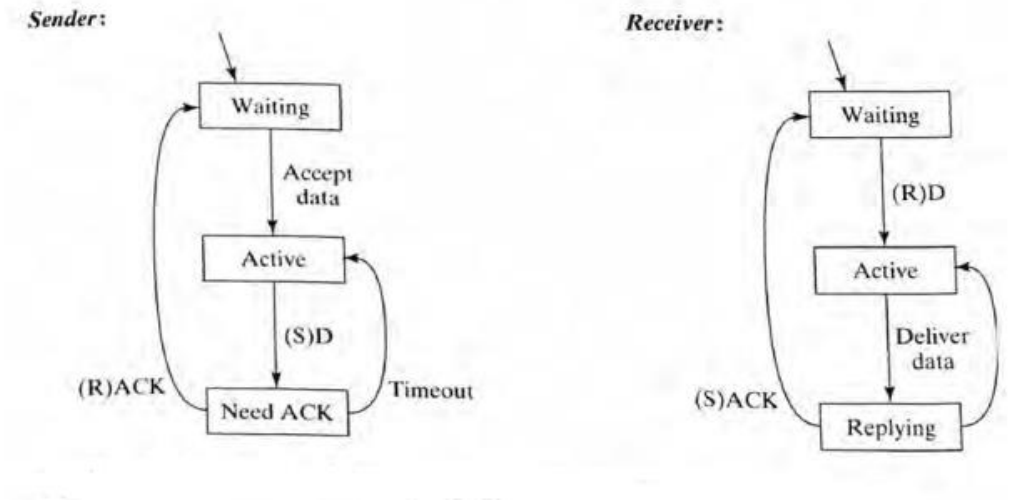
Model mesin finite state automata dari even pengiriman data dan even penerimaan dat, hal ini digambarkan dengan protokol yang sederhana. Dalam membangun model ini, diasumsikan bahwa ada proses dari tingkat lebih tinggi di satu sisi yang memanggil pengirim data ketika data dari sumber untuk dikiirim dan proses ke tingkat yang lebih tinggi pada sisi lain yang akan diberitahukan kepada penerima setiap kali data sampai tujuan. Dalam hal ini bahwa pengirim akan mempertahankan antrian FIFO, yaitu data yang pertama itu yang pertama diterimanya) itu akan menghapus dan mengirim pesan.

2.3.2 Protocol Stop-and-Wait

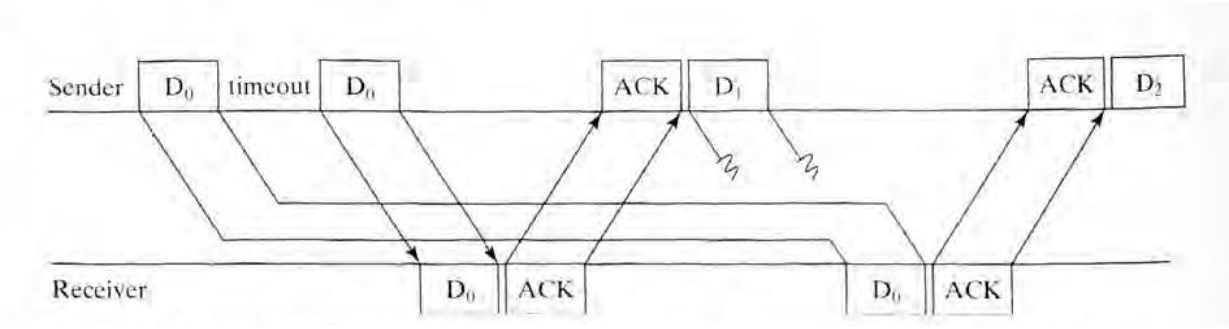
Protokol Automatic Repear reQuest (ARQ) telah dibuat untuk menyelesaikan dua masalah yaitu bagaimana pengiriman data dan penerimaan data berjalan dengan sukses. Dalam protokol ARQ, penerima berkomunikasi kembali ke pengirim ketika data dikirim dari sumber maka ketika data sampai ke tujuan system memberitahukan bahwa data sudah diterima (ACK), ketika data dikirim pengirim menunggu pernyataan balasanan apakah data diterima atau belum diterima dan demikian sebaliknya, jika belum diterima maka dari pihak penerima minta untuk dikirim ulang (request to sent)



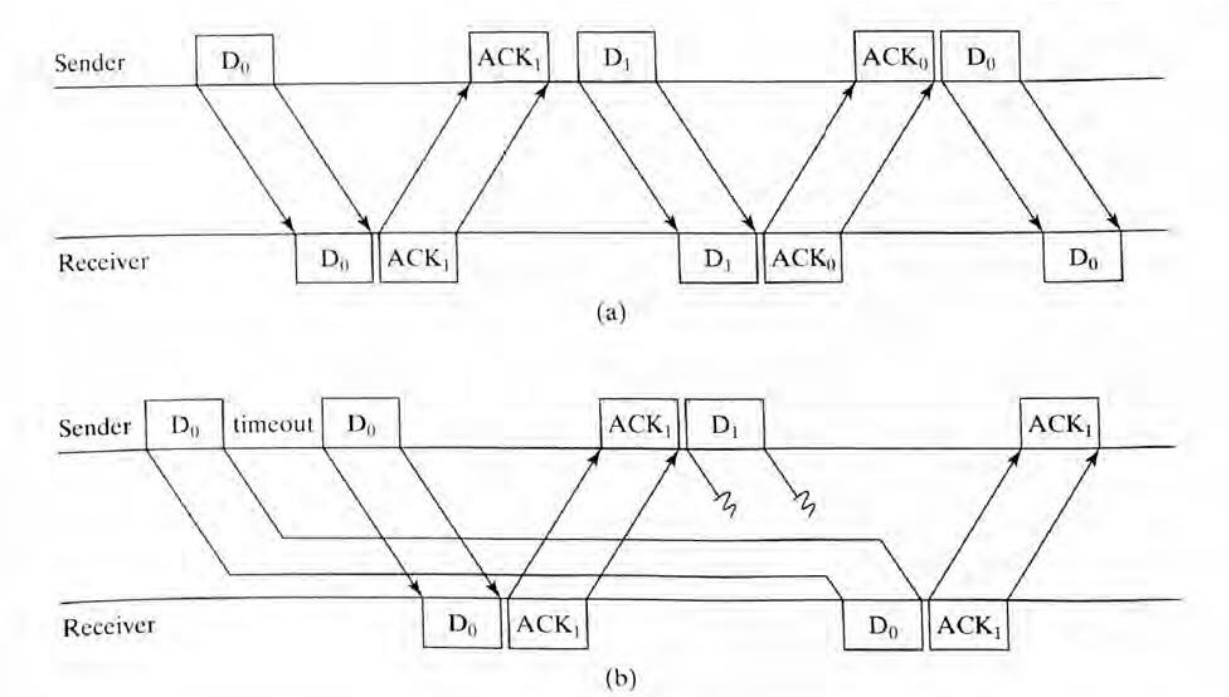
Gambar 2.10: Protokol ARQ



Gambar 2.11: Pengirim dan Penerima protocol Stop dan Wait

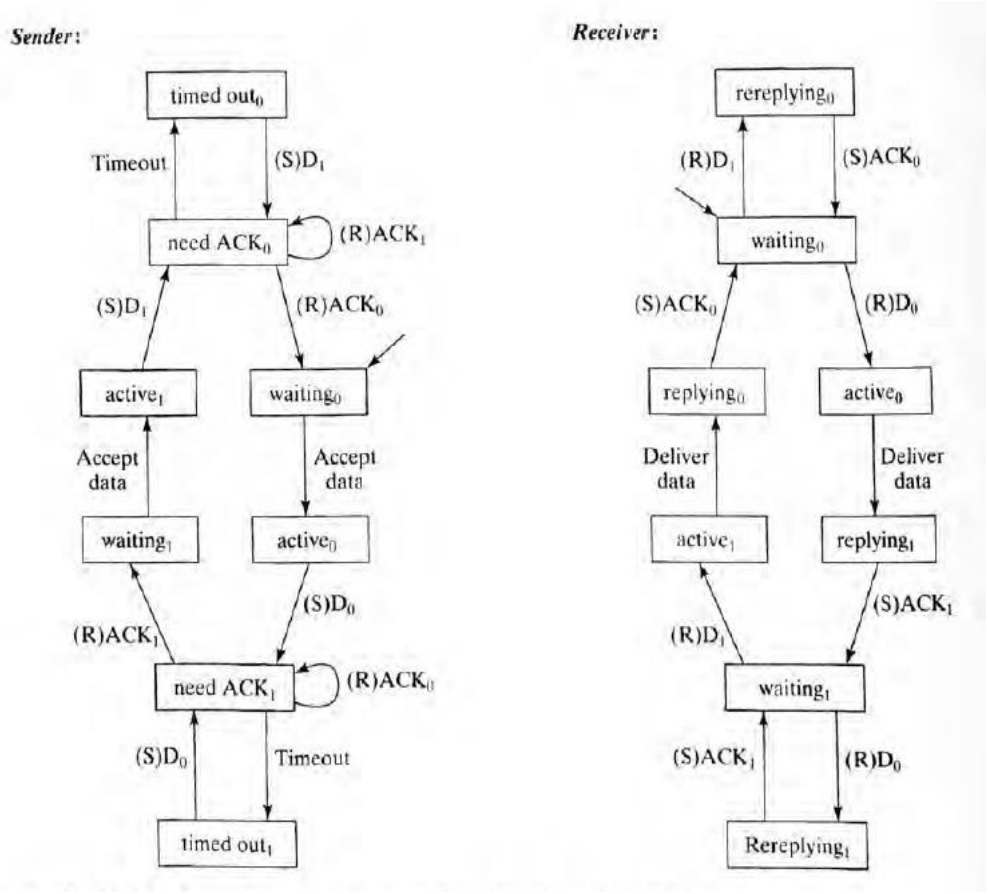


Gambar 2.12: Pengiriman ulang akibat data telah hilang

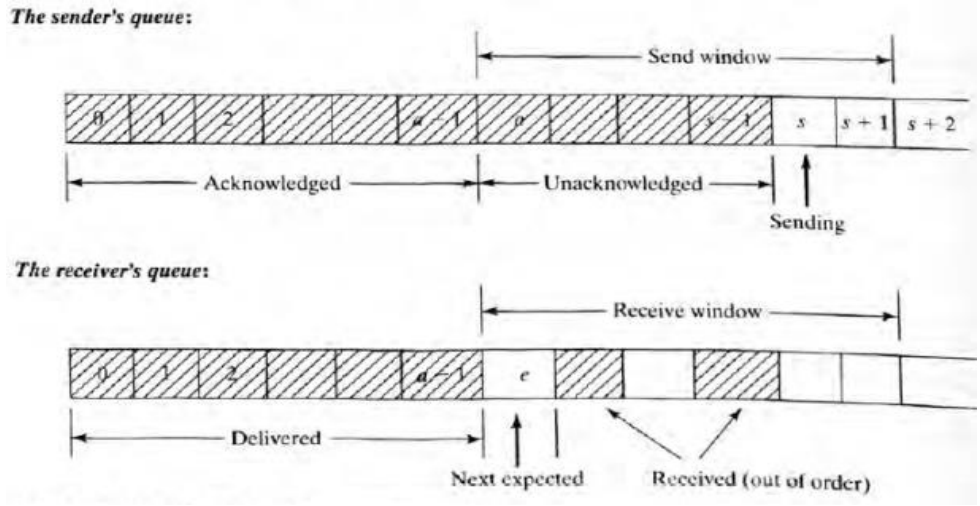


Gambar 2.13: Protokol Bit Bergantian

2.3.3 Sliding Window Protocol



Gambar 2.14: Model pengirim dan penerima untuk protokol Bit pengganti.



Gambar 2.15: Protokol Sliding Window

Koneksi berlangsung melalui pemeriksaan status selama masa pakainya.

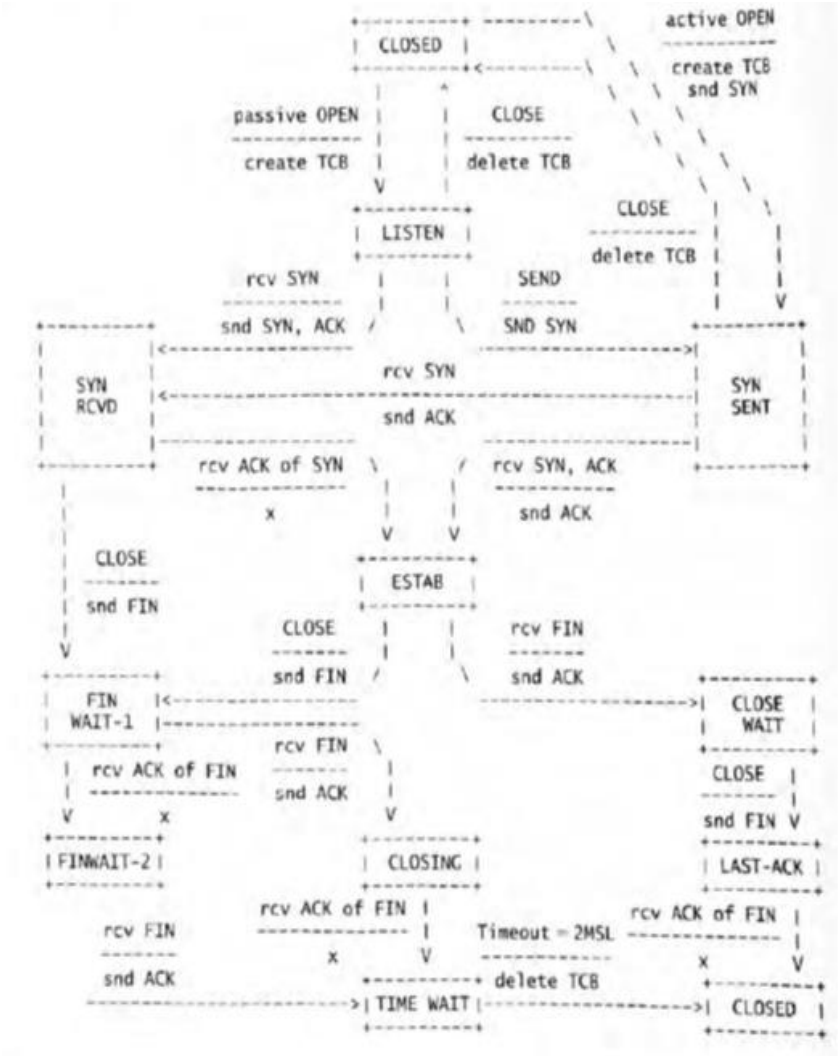
State-stete tersebut adalah:

states : LISTEN, SYN-SENT, SYN-RECEIVED, ESTABLISHED, FIN-WAIT-1,

FIN-WAIT-2, CLOSE-WAIT, CLOSING, LAST-ACK, TIME-WAIT,State Close. Close mewakili keadaan ketika tidak ada TCB [Blok Kontrol Transmisi], dan oleh karena itu, tidak ada koneksi. Secara singkat arti dari state adalah:

Tabel 2.2: Proses konelksi.

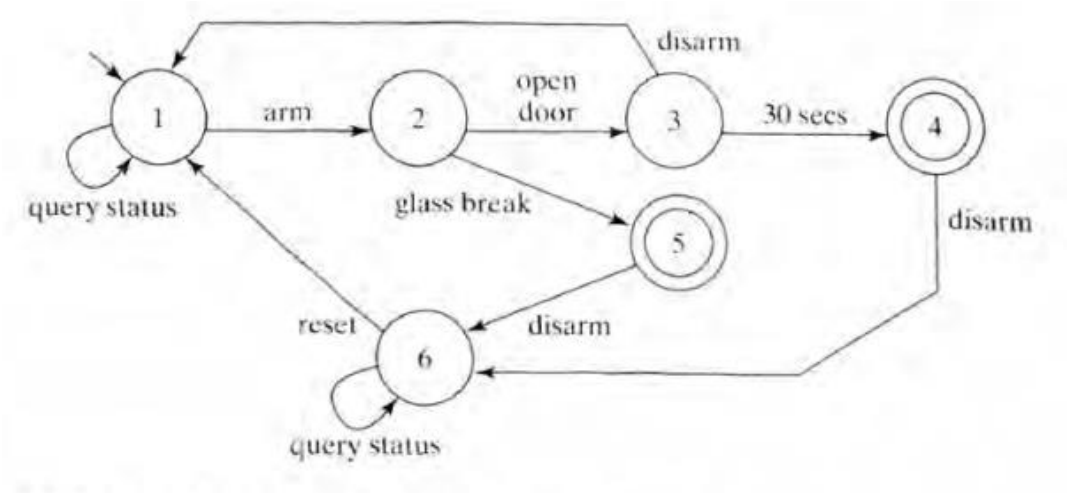
State	Keterangan
LISTEN	mewakili menunggu permintaan koneksi dari TCP dan port jarak jauh.
SYN-SENT	mewakili menunggu permintaan koneksi yang cocok setelah mengirim permintaan koneks
SYN-RECEIVED	- mewakili menunggu konfirmasi permintaan koneksi permintaan
ESTABLISHED	merupakan koneksi terbuka, data yang diterima dapat dikirimkan kepada pengguna. Keadaan normal untuk fase transfer data dari koneksi.
FIN-WAIT-1,	mewakili menunggu permintaan penghentian koneksi dari TCP jarak jauh, atau pengakuan atas permintaan penghentian koneksi yang sebelumnya dikirim.
FIN-WAIT-2	mewakili menunggu permintaan penghentian koneksi dari TCP jarak jauh
CLOSE-WAIT	mewakili menunggu permintaan penghentian koneksi dari pengguna loeal.
CLOSING	mewakili menunggu permintaan penghentian koneksi yang diketahui
LAST-ACK	mewakili menunggu untuk mengetahui penghentian koneksi pencarian sebelumnya dikirim ke TCP jarak jauh (yang mencakup pengakuan permintaan pemutusan koneksi
TIME-WAIT	mewakili waktu tunggu yang cukup untuk memastikan TCP jarak jauh menerima kembali pengakuan permintaan penghentian koneksinya.
Close	tidak menunjukkan kondisi koneksi sama sekali.



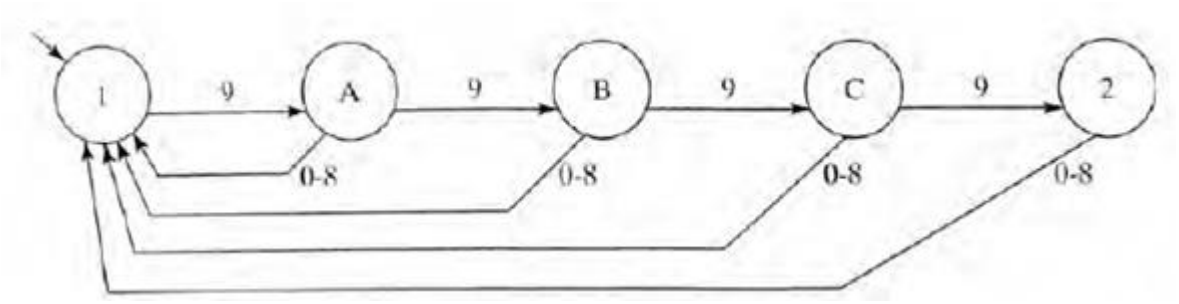
Gambar 2.16: Konsep Finite Automata Model Tranducer TCP

2.3.4 Sistem Keamanan Phisik

Konsep Automata digunakan untuk keamana kantor dan gedung



Gambar 2.17: Konsep Sistem keamanan dengan model Automata



Gambar 2.18: Sistem keamanam

Demikian pula penelitian untuk kecerdasan yang dilakukan oleh Sumarno Sumarno1, Danang Tri Prasetiyo, (2020) Lovebirds type identification designing based on color using automaton theory

RESEARCH PARKS
INTERNATIONAL JOURNAL ON ORANGE TECHNOLOGIES (IJOT) e-ISSN: 2615-814
Volume: 01 Issue: 02 | Nov-Dec 2019 www.researchparks.org p-ISSN: 2615-7071

Lovebirds type identification designing based on color using automaton theory

Sumarno Sumarno[1], Danang Tri Prasetiyo[2]

[1,2]*Informatics, Universitas Muhammadiyah Sidoarjo, Indonesia*

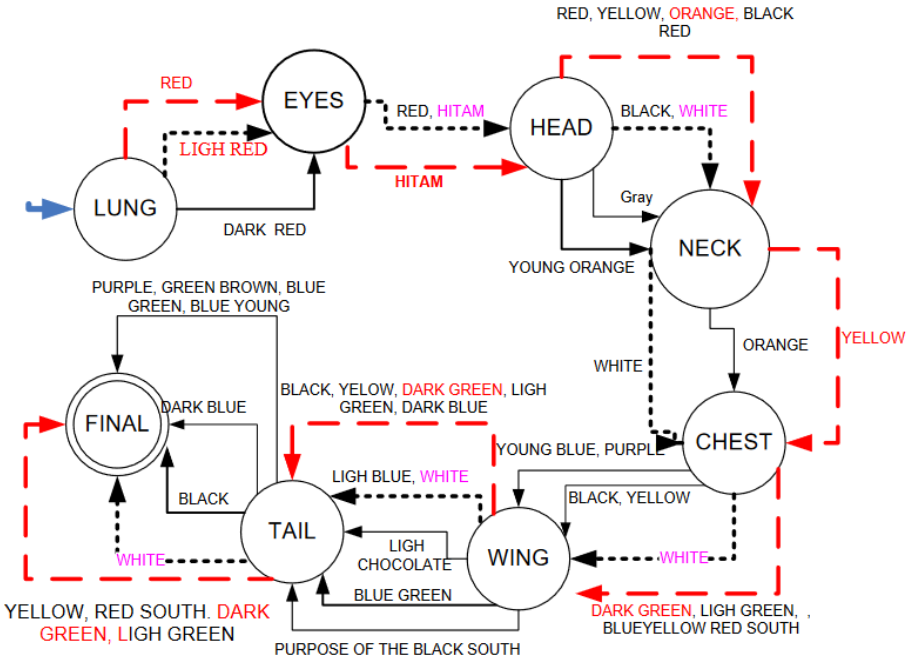
E-mail: sumarno@umsida.ac.id, danangtp96@gmail.com

Abstract - *Lovebirds are a unique and legendary bird species that can attract most bird lovers in Indonesia. The colors and types are also varied, ranging from valves, non-valves, colorful feathers starting from the lungs, eyes, chest, tail wings, and so forth. In addition to a variety of colors, the birds singing is also another attraction for bird lovers. But for the general public as well as beginner Lovebirds hobbyists, it is difficult to know the type of each Lovebirds they meet or have. This is due to the lack of knowledge they have about Lovebirds, so we need a system that can help them in solving problems to find out the name of the species on a specific basis.*

Expert systems are computer programs that simulate the reasoning of an expert with expertise in a particular area of knowledge. Expert systems work based on theory, rules, and knowledge base. The theory is the necessary foundation in making an expert system as a substitute for an expert's logic mindset. Automaton theory of intelligent abstract modeling recognition machine is one theory that can work on a system to resolve uncertainty according to predetermined rules, especially reading the input of words or context-free language, and then can conclude whether the writing is following the state of the regulation set in the programming language. The results of this study resulted in an expert system application to identify Lovebirds based on the color of feathers found in the body of the lovebird using finite-state Automata.

Key Words: *Expert System, Lovebirds, Automaton Theory.*

Gambar 3.19: Penggunaan teori Automata untuk Deteksi Jenis burung



Gamabar 2.20: Model Sistem Pakar.

Bab 3

Ekpresi Regular

3.1 Ekpresi Regular

Dalam bab ini, terdapat dua metode alternatif untuk menggambarkan bahasa regular, ekspresi reguler dan tata bahasa reguler. Ekspresi reguler, yang bentuknya dalam aritmatika yang sudah dikenal yaitu ekspresi,Tujuan dari bab ini adalah untuk mengeksplorasi kesetaraan dari tiga mode ini untuk menggambarkan bahasa reguler. Konstruksinya yang membuat konversi dari satu bentuk ke bentuk lain.

Karena setiap representasi dari bahasa reguler sepenuhnya dapat dikonversi untuk yang lain, kita dapat memilih mana yang paling sederhana, Menurut definisi, bahasa yang dapat didefinisikan dengan regular ekpresi maka bahasa itu disebut bahasa regular, Salah satu cara untuk menggambarkan bahasa reguler adalah melalui notasi regular ekspresi. Notasi ini melibatkan kombinasi string simbol dari beberapa alfabet , tanda kurung, dan operator +, •, dan . Sebagaimana bab sebelumnya membahaas tentang bahasa L dalam Bab 1 dengan simbul,

$$L1 = \{xn \text{ untuk } n = 1, 2, 3, \ldots \ldots,\}$$

Mari kita pertimbangkan kembali bahasa L, sebagai berikut

L4,={Ɛ, x, xx, xxx, xxxx,……..,……..}, **dapat ditulis sbb;**

Let S = {x}. Then L4 = **S***

L4 = {x}*

Dengan menggunakan notasi Kleene star maka menjadi x*, sehingga,

L4 = language {x}*, bahasa L4 ini didefinisikan dengan ekpresi regular x*

Sebagai contoh

L = {a, ab, abb, abbb, abbbb,……., ... }

L = language (a b*)

L = language (ab*)

Sehingga regular ekpresinya dari bahasa L adalah ab*

Mendefinisikan regular ekpresi menjadi bahasa regular, contoh regular ekspresi (ab)*, mala bahasa Lnya adalah

L (ab)* = {λ or ab or abab or ababab,……..,……,……}

Jika kita ingin mendefinisikan bahasa L yang lain, misalkan

L = language (xx*)

L = language (x+)

Semua kata x dengan pangkat + itu artinya himpunan kata yang terdiri dari huruf x tidak termasuk nullstring

Contoh

Bahasa L, dapat didefinisikan dengan salah satu ekspresi di bawah ini:

XX*, XX*X*, X*XX*, x x* x*x+ x**x*xx*, ……..

Ingat x* selalu bisa menjadi null string.

Contoh

Bahasa yang didefinisikan oleh ekspresi:

ab*a

bahasa dari regular ekspresinya L = language {ab*a}, yaitu bahasa yang diawali dan diakhir dengan huruf a, dan diantaranya terdapat beberapa huruf b, hingga tak terbatas, termasuk didalamnya himpunan kosong, sehingga bahasa itu adalah,

L = {{**aa,** aba, abba, abbba, abbbba ... .}

Bahasa dari ekspresi **a*b*, maka bahasa regularnya adalah**

language (a*b*) = { λ, a, b, aa, ab, bb, aaa, aab, abb, bbb, aaaa, . . . }

Contoh-contoh regular ekspresi dan bahasa regular

1) Consider the language T defined over the alphabet Σ = {a, b, c}

T = {a, c, ab, cb, abb, cbb, abbb, cbbb, abbbb, cbbbb ... }

T = language ((a + c)b*)

2) Kita perhatikan bahasa berhingga L

L = {aaa, aab, aba, abb, baa, bab, bba, bbb,}

L = language ((a + b)(a + b)(a + b))

3) Ditujukkan regular ekpresi (a + b)

L = language (a + b)*

L ={ Ɛ, a,b, aa, bb,ab,ba,baa,abb,aabb,bba, abab,bbba,aaab,ababa,…….,…}

Simbol penentu bahasa yang akan DISKUSIKAN ADLAH reguler ekspresi. kita akan mendefinisikan istilah ekspresi reguler itu sendiri secara rekursif. Bahasa itu yang yang dedefinikan dengan ekspresi reguler ini disebut bahasa reguler Dan dapat juga dikatakan atau didefinisikan yang terbatas.

Mari kita pertimbangkan kembali bahasa L4 dari.

$$L_4 = \{ \varepsilon \quad x \quad xx \quad xxx \quad xxxx \ . \ . \ . \}$$

Dalam hal ini kami menyajikan satu metode untuk menunjukkan himpunan set ini sebagai akhir dari himpunan yang lebih kecil.

$$Let\ S = \{X\}. Then\ L^4 = S^8$$

Dapat ditulis dengan

$$L_4 = \{x\}^*$$

Kami sekarang memperkenalkan penggunaan bintang Kleene yang diterapkan bukan pada set tetapi secara langsung ke huruf x dan ditulis sebagai superskrip seolah-olah itu eksponen.

$$\mathbf{x^*}$$

Ekspresi sederhana x* akan digunakan untuk menunjukkan beberapa urutan x's

(mungkin tidak ada sama sekali).

$$\mathbf{x^*} = \varepsilon \text{ or } x \text{ or } x^2 \text{ or } x^3 \text{ or } x^4 \ldots$$
$$= x^n \text{ for some } n = 0 \;\; 1 \;\; 2 \;\; 3 \;\; 4 \ldots$$

Kita dapat menganggap pangkat bintang sebagai kekuatan yang tidak diketahui atau kekuatan yang tidak ditentukan. x* adalah singkatan dari string x, tapi kami tidak menentukan berapa banyak. Itu berdiri untuk setiap string x di dalam bahasa L4 .

Operator bintang yang diterapkan pada huruf dianalogikan dengan operator bintang yang diterapkan untuk satu set. rangkaian Ini mewakili arbitrer dari pengganti arti word (mungkin tidak ada sama sekali). Notasi ini dapat digunakan untuk membantu kita mendefinisikan bahasa dengan menulis

$$L_4 = language\ (x^*)$$

Karena x* adalah sembarang string dari x, maka L4 adalah himpunan semua string yang mungkin dari x dengan panjang berapa pun (termasuk Ɛ). Kita tidak boleh bingung dengan notasi x*, yang merupakan simbol yang mendefinisikan bahasa, dengan L4 ,yang merupakan nama yang kami berikan untuk bahasa tertentu. Inilah sebabnya kami menggunakan kata "bahasa" dalam persamaan. Kami akan segera memberi nama pada dunia di mana simbol x* ini hidup tetapi belum sepenuhnya. Misalkan kita ingin jelaskan bahasa L di atas alfabet Ʃ = {a,b} di mana

$$L = \{\, a \quad ab \quad abb \quad abbb \quad abbbb \quad \ldots \}$$

Kita dapat meringkas bahasa ini dengan frasa bahasa Inggris "all words dari

bentuk satu a diikuti oleh sejumlah b (mungkin tidak ada b sama sekali.)"

kami dapat menulis:

$$L = language\ (a\ b^*)$$

Artinya jelas: Ini adalah bahasa di mana kata-katanya adalah gabungan dari diawali dengan huruf a dengan beberapa atau tanpa huruf b (yaitu b*). Apakah kita menempatkan spasi di dalam ab* atau tidak hanya untuk kejelasan membaca; itu tidak berubah set string yang diwakilinya. Tidak ada string yang dapat berisi yang kosong kecuali yang kosong adalah karakter dalam alfabet 1. Jika kita ingin yang kosong ada di alfabet, kita biasanya memperkenalkan beberapa simbol khusus untuk mewakili mereka, sebagai kosong itu sendiri tidak terlihat oleh mata telanjang. Alasan untuk mengosongkan antara a dan b* pada produk di atas adalah untuk menekankan bahwa operator pangkat bintang adalah diterapkan pada b saja. Kami sekarang telah menggunakan huruf tebal tanpa bintang sebagai juga dengan bintang. Kita dapat menerapkan bintang Kleene ke string ab jika kita mau, sebagai berikut:

$$\mathbf{(ab)^*} = \Lambda \quad \text{or} \quad ab \quad \text{or} \quad abab \quad \text{or} \quad ababab \ \ldots$$

Tanda kurung bukan huruf dalam alfabet bahasa ini, jadi bisa

digunakan untuk menunjukkan pemfaktoran tanpa sengaja mengubah kata-kata. Sejak

bintang mewakili semacam eksponensial, kami menggunakannya sebagai kekuatan yang digunakan dalam aljabar, di mana dengan pemahaman universal ekspresi xy2 berarti x(y2), bukan

(Xy)2.

Jika kita ingin mendefinisikan bahasa L, dengan cara ini, kita dapat menulis

$$L_1 = language\ (x\ x^*)$$

Ini berarti bahwa kita memulai setiap kata dari L, dengan menuliskan x dan kemudian kita ikuti dengan beberapa string x (yang mungkin tidak ada lagi x sama sekali). Atau kita dapat menggunakan notasi + dan menulisnya

$$L_1 = language\ (x^*)$$

artinya semua kata dalam bentuk x ke suatu pangkat positif (yaitu, bukan x0 = Ɛ).

Notasi + adalah kenyamanan tetapi tidak penting karena kita dapat mengatakan

hal yang sama dengan *'s saja.

Contoh

Bahasa L, dapat didefinisikan dengan salah satu ekspresi di bawah ini:

$$\mathbf{xx^*} \quad \mathbf{x^+} \quad \mathbf{xx^*x^*} \quad \mathbf{x^*xx^*} \quad \mathbf{x^+x^*} \quad \mathbf{x^*x^+} \quad \mathbf{x^*x^*x^*xx^*}$$

Ingat x* selalu bisa menjadi Ɛ.

Contoh

Bahasa yang ditentukan oleh ekspresi

$$\mathbf{ab^*a}$$

adalah himpunan semua string a dan b yang memiliki setidaknya dua huruf, yang dimulai dan diakhiri dengan a, dan itu tidak memiliki apa-apa selain b di dalam (jika ada).

$$\text{language } (\mathbf{ab^*a}) = \{aa \quad aba \quad abba \quad abbba \quad abbbba \quad \ldots\}$$

Akan menjadi kesalahan kecil untuk mengatakan bahwa bahasa ini adalah kumpulan dari semuanya kata-kata yang dimulai dan diakhiri dengan a dan hanya memiliki b di antaranya, karena deskripsi ini mungkin juga berlaku untuk kata "a", tergantung pada bagaimana itu ditafsirkan. Secara Simbolis menghilangkan ambiguitasnya.

Contoh

Bahasa ekspresi sbb ;

$$\mathbf{a^*b^*}$$

berisi semua string a dan b di mana semua a (jika ada) muncul sebelumnya semua b (jika ada).

$$\text{language } (\mathbf{a^*b^*}) = \{\Lambda\ a\ b\ aa\ ab\ bb\ aaa\ aab\ abb\ bbb\ aaaa\ \ldots\}$$

Perhatikan bahwa ba dan aba tidak dalam bahasa ini. Perhatikan juga bahwa ada kebutuhan tidak sama jumlah a dan b. Di sini kita sekali lagi harus sangat berhati-hati untuk mengamati nya

$$\mathbf{a^*b^*} \neq \mathbf{(ab)^*}$$

karena bahasa yang didefinisikan oleh ekspresi di sebelah kanan mengandung kata abab, yang bahasanya tidak didefinisikan oleh ekspresi di sebelah kiri. Ini memperingatkan kita agar tidak menganggap * sebagai eksponen aljabar normal. Bahasa yang didefinisikan oleh ekspresi a*b*a* mengandung kata baa karena itu dimulai dengan nol a diikuti oleh satu b diikuti oleh dua a.

Contoh

Ekspresi berikut keduanya mendefinisikan bahasa L2 = {xodd}

$$\mathbf{x(xx)^*} \ \ or \ \ \mathbf{(xx)^*x}$$

tapi ekspresinya

$$\mathbf{x^*xx^*}$$

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

Kami sekarang memperkenalkan penggunaan lain untuk tanda plus. Dengan ekspresi x + y di mana x dan y adalah string karakter dari alfabet, yang kami maksud adalah "salah satu" x atau y". Ini berarti x + y menawarkan pilihan, sama seperti x*

melakukan. Perawatan harus diambil agar tidak membingungkan ini dengan + sebagai eksponen.

Contoh

Pertimbangkan bahasa T yang didefinisikan over alfabet ∑ {a, b, c}

T = {a c ab cb abb cbb abbb cbbb abbbb cbbbb ... }, Semua kata dalam T dimulai dengan a atau a c dan kemudian diikuti oleh beberapa jumlah b. Secara simbolis, kita dapat menulis ini sebagai

T = bahasa ((a + c)b*) = bahasa (baik a atau c lalu beberapa b)

Kita harus, mengatakan "beberapa atau tidak ada b". Kami sering menjatuhkan opsi nol karena melelahkan. Kami membiarkan kata "beberapa" selalu berarti "beberapa atau tidak," dan ketika kami bermaksud "beberapa angka positif", kami mengatakan itu. Kami mengatakan bahwa ekspresi (a + c)b* mendefinisikan bahasa sebagai berikut: nalar. Setiap + dan * meminta kita untuk membuat pilihan. Untuk setiap * atau ± digunakan sebagai superskrip kita harus memilih beberapa faktor yang mendukungnya. Untuk satu sama lain + kita harus memutuskan apakah akan memilih ekspresi sisi kanan atau ekspresi sisi kiri. Untuk setiap rangkaian pilihan, kami telah menghasilkan tertentu rangkaian. Himpunan semua string yang dapat dihasilkan oleh metode ini adalah bahasa dari ekspresi. Dalam contoh

$$\mathbf{(a + c)b^*}$$

kita harus memilih a atau c untuk huruf pertama dan kemudian kita memilih berapa b singkatan dari b*. Setiap rangkaian pilihan adalah sebuah kata. Jika dari (a + c) kita pilih c dan kita pilih b* artinya bbb, kita punya kata cbbb.

Contoh

Sekarang mari kita perhatikan bahasa berhingga L yang berisi semua string a's dan b panjangnya tepat tiga

$$L = \{aaa \quad aab \quad aba \quad abb \quad baa \quad bab \quad bba \quad bbb\}$$

Huruf pertama dari setiap kata dalam L adalah a atau a b. Huruf kedua dari setiap kata dalam L adalah a atau a b. Huruf ketiga dari setiap kata dalam L adalah baik a atau b. Jadi kita bisa menulis

$$L = \text{language } (\mathbf{(a + b)(a + b)(a + b)})$$

Singkatnya dapat ditulis

$$L = \text{language } (\mathbf{(a + b)^3})$$

Jika kita ingin mendefinisikan himpunan ketujuh string huruf a dan b, kita bisa menulis (a + b)7 . Secara umum, jika kita ingin merujuk ke himpunan semua kemungkinan string a dan b dengan panjang berapa pun dapat kita tulis

$$\mathbf{(a + b)^*}$$

adalah himpunan semua kemungkinan string huruf dari alfabet ϵ {a, b} termasuk string nol. Ini adalah ekspresi reguler yang sangat penting dan kami sering menggunakannya. Sekali lagi ungkapan ini mewakili bahasa. Jika kita memutuskan bahwa * berarti 5, maka

$$\mathbf{(a + b)^*}$$

Diberikan sbb

$$\mathbf{(a + b)^5 = (a+b)(a+b)(a+b)(a+b)(a+b)}$$

sekarang harus membuat lima pilihan lagi: a atau b untuk huruf pertama, baik a atau b untuk huruf kedua ..... Ini adalah notasi yang sangat kuat. Kami dapat menggambarkan semua kata yang dimulai dengan huruf a secara sederhana sebagai:

$$a(a + b)^*$$

yaitu, pertama a, lalu apa saja (pilihan sebanyak yang kita inginkan dari salah satu huruf a atau b).

Semua kata yang dimulai dengan a dan diakhiri dengan b dapat didefinisikan dengan ekspresi

$$a(a + b)^*b = a \text{ (string panjangnya bebas) } b$$

Contoh

Mari kita perhatikan bahasa yang didefinisikan oleh ekspresi

$$(a + b)^*a(a + b)^*$$

Pada awalnya kita memiliki

$$(a + b)^*$$

yang berarti apa sajabeerapa a atau beberapa b termasuk kata kosong, yaitu apa saja string a dan b, lalu muncul a, lalu apa saja. Semua mengatakan, bahasa adalah himpunan semua kata di atas abjad Y- = {a, b} yang memiliki a di suatu tempat. Satu satunya kata yang tertinggal adalah kata-kata yang hanya memiliki beberapa b dan kata Ɛ Misalnya, kata abbaab dapat dianggap dari bentuk ini dalam tiga cara:

$$(Ɛ) \text{ a (bbaab) atau (abb) a (ab) atau (abba) a (b)}$$

Contoh

Bahasa semua kata yang memiliki setidaknya dua a dapat dijelaskan oleh ekspresi

$$(a + b)^*a(a + b)^*a(a + b)^*$$

= (beberapa awal)(pertama penting a)(beberapa tengah)(kedua penting a)(akhirnya)

di mana bagian arbitrer dapat memiliki a (atau b) sebanyak yang mereka inginkan. EDalam tiga contoh terakhir kami telah menggunakan notasi (a + b)* sebagai faktor berarti "setiap substring yang mungkin," seperti yang telah kita lihat mewakili bahasa dari semua kata. Dalam pengertian ini, ekspresi (a + b)* adalah suatu kata yang bebas a, b, atau a kemudian b. atau disebut yang mana saja.

Contoh

Ekspresi lain yang menunjukkan semua kata dengan setidaknya dua a adalah:

$$b^*ab^*a(a + b)^*$$

Kami memindai melalui beberapa hutan b (atau tidak ada b) sampai kami menemukan a pertama, lalu lebih banyak b (atau tidak ada b), lalu yang kedua a, lalu kita selesaikan dengan apa saja. Di set ini adalah abbbabb dan aaaaa.

Kita dapat menulisnya :

$$(a + b)^*a(a + b)^*a(a + b)^* = b^*ab^*a(a + b)^*$$

di mana dengan tanda sama dengan kami tidak berarti bahwa ekspresi ini sama secara aljabar dengan cara yang sama seperti

$$x+x=2x$$

tetapi mereka sama karena mereka menggambarkan item yang sama, Kita bisa menulis

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

bahasa ((a + b)*a(a + b)*a(a + b)*)

= bahasa (b*ab*a (a + b)*)

= semua kata dengan setidaknya dua a.

berhati-hati tentang hal ini, kami mengatakan bahwa dua ekspresi reguler adalah ekuivalen jika mereka menggambarkan bahasa yang sama.

Contoh

Bahasa adalah semua kata yang memiliki setidaknya satu a dan setidaknya satu b adalah agak lebih rumit. Jika kita menulis

$$(a + b)*a(a + b)* b(a + b)*$$

kita kemudian mensyaratkan bahwa a mendahului a b dalam kata. Kata-kata seperti ba dan bbaaaa tidak termasuk dalam set ini. Karena, bagaimanapun, kita juga tahu itu a datang sebelum b atau b datang sebelum a, kita bisa mendefinisikan ini ditetapkan oleh ekspresi:

$$(a+b)*a(a+b)*b(a+b)* + (a+b)*b(a+b)*a(a+b)*$$

Di sini kita masih menggunakan tanda plus dalam pengertian umum disjungsi (atau).

Kami mengambil gabungan dua set, tetapi lebih tepat untuk memikirkan tanda + sebagai alternatif dalam membentuk kata.

Ekspresi sederhana yang mendefinisikan bahasa yang sama.

$$(a + b)*a(a + b)*b(a + b)*$$

adalah kata-kata dari bentuk beberapa b diikuti oleh beberapa a, maka itu akan menjadi cukup untuk menambahkan pengecualian khusus ini ke dalam set. Pengecualian ini adalah semua ditentukan oleh ekspresi reguler:

$$bb*aa*$$

Bahasa semua kata di atas alfabet $\sum = \{a, b\}$ yang mengandung keduanya a dan a b karena itu juga didefinisikan oleh ekspresi:

$$(a + b)*a(a + b)*b(a + b)* + bb*aa*$$

Perhatikan bahwa perlu untuk menulis bb*aa* karena b*a* akan menerima kata-kata

kami tidak ingin, seperti aaa.

Ekspresi yang mendefinisikan bahasa ini tidak dapat diperlakukan seperti simbol aljabar. Kami telah menunjukkan bahwa

$$(a + b)*a(a + b)*b(a + b)* + (a + b)*b(a + b)*a(a + b)* = (a + b)*a(a + b)*b(a + b)* + bb*aa*$$

Suku pertama pada kedua ruas persamaan ini sama, tetapi jika kita abaikan kita dapatkan ekspresi

$$(a+b)*b(a+b)*a(a+b)* = bb*aa*$$

yang salah, karena sisi kiri termasuk kata aba, yang ekspresinya di sisi kanan tidak.Satu-satunya kata yang tidak mengandung a dan b di suatu tempatadalah kata-kata dari semua a, semua b, atau Ɛ. Ketika ini ditambahkan ke dalam bahasa,kita mendapatkan segalanya. Oleh karena itu, ekspresi reguler:

$$(a+b)*a(a+b)*b(a+b)* + bb*aa* + a* + b*$$

mendefinisikan semua kemungkinan string a dan b. Kata A termasuk dalam a* dan b*.

Kita kemudian dapat menulis:

$$(a + b)^* = (a+b)^*a(a+b)^*b(a+b)^* + bb^*aa^* + a^* + b^*$$

yang sama sekali bukan kesetaraan yang sangat jelas.

Contoh

Semua ekspresi dibawah ini mendefinisikan bahasa yang sama

$(a+b)^* = (a+b)^* + (a+b)^*$

$(a+b)^* = (a+b)^* (a+b)^*$

$(a+b)^* = a(a+b)^* + b(a+b)^* + A$

$(a+b)^* = (a+b)^* ab(a+b)^* + b^*a^*$

Definisi

Himpunan ekspresi reguler didefinisikan oleh aturan berikut:

Aturan 1 Setiap huruf dari 1 dapat dibuat menjadi ekspresi reguler dengan menuliskannya di tebal; Ɛ adalah ekspresi reguler.

Aturan 2 Jika r1 dan r2 adalah ekspresi reguler, maka demikian juga

$$(r1) \quad r1\ r\,2\ \ r1\ +\ r\,2\ \ r\,1^*$$

Aturan 3 Tidak ada yang lain adalah ekspresi reguler.

Kita bisa memasukkan tanda plus sebagai superskrip ri+ sebagai bagian dari definisi, tetapi karena kita tahu bahwa rl+ = rlrl*,.

Definisi

Jika S dan T adalah himpunan untaian huruf (apakah itu himpunan berhingga atau tak berhingga), kita mendefinisikan himpunan hasilkali menjadi ST = {semua kombinasi string dari S yang digabung dengan string dari T }

Contoh

Jika S = {a aa aaa} dan T = {bb bbb}, maka ST = {abb abbb aabb aabbb aaabb aaabbb}

Contoh

Jika S = {a, bb, bab}, dan T = {a, ab}, maka ST = {aa aab bba bbab baba babab}

Definisi

Aturan berikut mendefinisikan bahasa yang terkait dengan ekspresi reguler Aturan 1: Bahasa yang terkait dengan ekspresi reguler yang hanya a satu huruf adalah kata satu huruf itu sendiri dan bahasa yang terkait dengan Ɛ hanyalah {Ɛ}, bahasa itu adalah satu kata. Aturan 2 Jika r, adalah ekspresi reguler yang terkait dengan bahasa L, dan r2 adalah ekspresi reguler yang terkait dengan bahasa L2 maka,

(i) Ekspresi reguler (rl) (r 2) dikaitkan dengan bahasa L , kali L 2.

$$\text{bahasa } (r, r2\ ) = L1\ L\ 2$$

(ii) Ekspresi reguler r, + r 2 dikaitkan dengan bahasa yang dibentuk dengan penyatuan himpunan L 1 dan L 2 .

$$\text{bahasa } (rl + r\ 2) = L, + L2$$

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

(iii) Bahasa yang terkait dengan ekspresi reguler (rl)* adalah LI*,Kleene cloususre dari himpunan LI sebagai himpunan kata.

$$\text{bahasa (rl*)} = L1^*$$

Sekali lagi kumpulan aturan ini membuktikan secara rekursif bahwa ada beberapa bahasa yang terkait dengan ekspresi reguler. Saat membangun reguler ekspresi dari aturan, kami secara bersamaan membangun yang sesuai bahasa. tampaknya Aturan menunjukkan kepada kita bagaimana kita dapat menafsirkan ekspresi reguler sebagai bahasa, tetapi mereka tidak benar-benar memberitahu pada kita bagaimana memahami bahasa. Maksudnya adalah jika kami menerapkan aturan di atas ke ekspresi reguler

$$(a + b)^*a(a + b)^*b(a + b)^* + bb^*aa$$

Maka kita dapat katakana bahawa ekpresi regular diatas menunjukkan bahasa yang terdiri dari factor atau huruf a dn b, sehingga kata miniman kalau kita lihat sisi ruas kanan dari plus kata minimannya adalah baa dan sebelah kiri adalah ab.

Korespondensi antara ekspresi reguler dan bahasa ini terbuka dua pertanyaan lainnya. Kami telah melihat contoh di mana sangat berbeda ekspresi reguler akhirnya menggambarkan bahasa yang sama. Apakah ada cara? memberitahu kapan ini terjadi? Dengan "cara" yang kami maksud, tentu saja, sebuah algoritma. Kami menyajikan prosedur algoritmik untuk menentukan apakah atau tidak dua ekspresi reguler mendefinisikan bahasa yang sama. Pertanyaan mendasar lainnya adalah ini: Kami telah melihat bahwa setiap reguler ekspresi dikaitkan dengan beberapa bahasa; apakah benar juga bahwa setiap bahasa dapat dijelaskan dengan ekspresi reguler? Dalam teorema kami berikutnya, kami menunjukkan bahwa setiap bahasa yang terbatas dapat didefinisikan oleh ekspresi reguler. Situasi untuk bahasa dengan banyak kata berbeda. Kami membuktikan bahwa ada beberapa bahasa yang tidak dapat didefinisikan oleh ekspresi reguler apa pun. Mengenai pertanyaan pertama dan mungkin yang paling penting, pertanyaan tentang tidak memahami ekspresi reguler, kami tidak tahu. Sebelum kita dapat membangun sebuah algoritma untuk mendapatkan pemahaman kita harus memiliki beberapa definisi yang baik tentang apa artinya memahami. Kita mungkin berabad-abad lagi untuk bisa untuk melakukan itu, jika itu bisa dilakukan sama sekali.

3.2 Teorema

Jika L adalah bahasa yang terbatas (bahasa dengan hanya banyak kata yang terbatas), maka L dapat didefinisikan oleh ekspresi regular untuk membuat satu ekspresi reguler yang mendefinisikan bahasa L,

$$L = \{baa\ abbba\ bababa\}$$

Adalah

$$baa + abbba + bababa$$

Jika L = {aa ab ba bb}, mka ekspresi regulernya adalah

$$aa + ab + ba + bb = (a + b)(a + b)$$

Contoh

Ditunjukkan L {Ɛ x xx xxx xxxx xxxxx}

Ekspresi reguler yang kita dapatkan dari teorema adalah

$$Ɛ + x + xx + xxx + xxxx + xxxxx$$

Ekspresi reguler yang lebih elegan untuk bahasa ini adalah

$$(Ɛ + x)5$$

Tentu saja angka 5, secara tegas, bukan simbol hukum untuk ekspresi reguler meskipun kita semua mengerti artinya

$$(Ɛ + x)(Ɛ + x)(Ɛ + x)(Ɛ + x)(Ɛ + x)$$

Mari kita periksa beberapa ekspresi reguler dan lihat apakah kita cukup untuk memahami sesuatu tentang bahasa yang mereka wakili.

Contoh

Pertimbangkan ekspresi reguler di bawah ini:

E = (a + b)*a(a + b)* (a + A) (a + b)*a(a + b)*

= (terdiri dari factor a dan b, termasuk nullable) a (terdiri dari factor a dan b, termasuk nullable) [a atau tidak sama sekali] (terdiri dari factor a dan b, termasuk nullable) a (terdiri dari factor a dan b, termasuk nullable).

Satu fakta yang jelas adalah bahwa semua kata dalam bahasa E harus memiliki setidaknya dua a di dalamnya. Mari kita bagi tanda tambah tengah menjadi dua kasus: baik faktor tengah memberikan kontribusi a atau yang lain memberikan kontribusi A. Oleh karena itu,

E = (a+b)*a(a+b)*a(a+b)*a(a+b)* + (a + b)*a(a + b)* A (a + b)*a(a + b)*

Ini adalah penggunaan yang lebih rinci dari hukum distributif. Istilah pertama di atas dengan jelas mewakili semua kata yang memiliki setidaknya tiga a di dalamnya. Sebelum kita menganalisis istilah kedua mari kita membuat pengamatan bahwa

(a + b)* Ɛ (a + b)*

yang terjadi di tengah istilah kedua hanyalah cara lain untuk mengatakan "string apa pun" dan dapat diganti dengan ekspresi yang lebih langsung (a + b)* Ini akan mengurangi istilah kedua dari ekspresi menjadi

(a + b)*a(a + b)*a(a + b)*

yang telah kita lihat adalah ekspresi reguler yang mewakili semua kata yang memiliki setidaknya dua a di dalamnya. Oleh karena itu, bahasa yang terkait dengan E adalah gabungan semua string yang memiliki tiga atau lebih a dengan semua string yang memiliki dua atau lebih a. Tapi ketika semua string dengan tiga atau lebih a itu sendiri sudah menjadi string dengan dua atau lebih a's, seluruh bahasa ini hanya set kedua saja

Bahasa yang diasosiasikan dengan E tidak berbeda dengan bahasa yang diasosiasikan dengan

(a + b)*a(a + b)*a(a + b)*

Hal ini dimungkinkan dengan penerapan aturan berulang untuk membentuk ekspresi reguler untuk menghasilkan ekspresi di mana operator bintang diterapkan ke a subekspresi yang sudah memiliki bintang di dalamnya. Beberapa contohnya adalah:

(a + b*)* (aa + ab*)* ((a + bbba*) + ba*b)*

Dalam ekspresi pertama ini, * internal tidak menambahkan apa pun ke bahasa

(a + b*)* = (a + b)*

karena semua kemungkinan string a dan b dijelaskan oleh kedua ekspresi. Juga, sesuai dengan Teorema 1,

(a*)* = a*

Namun,

(aa + ab*)* +ý(aa + ab)*

karena bahasa untuk ekspresi di sebelah kiri termasuk kata abbabb, yang bahasa di sebelah kanan tidak. (Bahasa yang ditentukan oleh regular ekspresi di sebelah kanan tidak boleh mengandung kata apa pun dengan dobel b.)

Contoh

Pertimbangkan ekspresi reguler:

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

(a*b*)*

Bahasa yang didefinisikan oleh ekspresi ini adalah semua string yang dapat dibuat dari faktor bentuk a*b*, tetapi karena kedua huruf a dan tunggal huruf b adalah kata-kata dalam bentuk a*b*, bahasa ini berisi semua string beberapa a dan b. Itu tidak bisa berisi lebih dari segalanya, jadi

(a*b*)* = (a + b)*

3.3 Latihan Soal

1. Biarkan ri, r 2 , dan r 3 menjadi tiga ekspresi reguler. Tunjukkan bahwa bahasa diasosiasikan dengan (r, + r 2)r 3 adalah sama, dengan bahasa yang diasosiasikan dengan rjr 3 + r 2 r 3. Tunjukkan bahwa r 1 (r 2 + r 3) setara dengan r 1r 2 + r j r 3 . Ini akan sama dengan "membuktikan hukum distributif" untuk ekspresi reguler.Bangun ekspresi reguler yang mendefinisikan masing-masing bahasa berikut: di atas alfabet I = {a, b}.
2. Semua kata di mana a muncul tiga kali lipat, jika sama sekali. Ini berarti bahwa setiap rumpun a berisi 3 atau 6 atau 9 atau 12... a.
3. Semua kata yang mengandung setidaknya satu string s, s 2 s 3 atau s4
4. Semua kata yang mengandung tepat tiga b secara total.
5. Semua kata yang mengandung tepat dua b atau tepat tiga b, tidak lebih.
6. (i) Semua string yang diakhiri dengan huruf ganda. (ii) Semua string yang memiliki tepat satu huruf ganda di dalamnya.
7. Semua string di mana huruf b tidak pernah tiga kali lipat. Ini berarti tidak kata berisi substring bbb.
8. Semua kata di mana a tiga kali lipat atau b tiga kali lipat, tetapi tidak keduanya. Ini berartisetiap kata mengandung substring aaa atau substring bbb tetapi tidak keduanya.
9. Semua string yang tidak memiliki substring ab.(ii) Semua string yang tidak memiliki substring bba dan abb.
10. Semua string di mana jumlah total a habis dibagi tiga, seperti: sebagai aabababa.
11. (i) Semua string di mana setiap b yang terjadi ditemukan dalam rumpun bilangan ganjil pada suatu waktu, seperti abaabbbab. (ii) Semua string yang memiliki bilangan genap a dan bilangan ganjil dari b. (iii) Semua string yang memiliki bilangan ganjil a dan bilangan ganjil dari b.
12. Mari kita pertimbangkan kembali ekspresi reguler (a + b)*a(a + b)*b(a + b)*(i) Tunjukkan bahwa ini ekuivalen dengan (a + b)*ab(a + b)*dalam arti bahwa mereka mendefinisikan bahasa yang sama. (ii) Tunjukkan bahwa(a + b)*ab(a +b)* + b*a* = (a + b)* (iii) Tunjukkan bahwa(a + b)* ab[(a + b)*ab(a + b)* + b*a*] + b*a* = (a + b)* (iv) Apakah (iii) variasi terakhir dari tema ini atau ada lebih banyak binatang? tertinggal di gua ini?
13. Kami telah mendefinisikan produk dari dua set string secara umum. Jika kita terapkan ini untuk kasus di mana kedua faktor adalah himpunan yang sama, S = T, kita mendapatkan kuadrat, S2. Demikian pula kita dapat mendefinisikan S3, S4. Menunjukkan bahwa

    (i) S* = Ɛ + S + S' + S2 + S3 + S 4 + ...

    (ii) (ii) S+ = S + S1 + S2 + S3 + S4 + ...

14. Tunjukkan bahwa pasangan ekspresi reguler berikut mendefinisikan hal yang sama bahasa di atas alfabet ∑ = {a, b}.
    (i) (ab)*a dan a(ba)*
    (ii) a* + b)* dan (a + b)* (iii) (a* + b*)* dan (a + b)* 15. (i) Ɛ* dan Ɛ (ii) (a*b)*a* dan a*(ba*)*
    (iii) (a*bbb)*a* dan a*(bbba*)* 16. (i) ((a + bb)*aa)* dan Ɛ + (a + bb)*aa (ii) (aa)*(Ɛ + a) dan a* (iii) a(aa)*(Ɛ + a)b + b dan a*b
15. (i) Ɛ* dan Ɛ
    (ii) (a*b)*a* and a*(ba*)*
    (iii) (a*bbb)*a* dan a*(bbba*)*
16. (i) ((a + bb)*aa)* dan Ɛ + (a + bb)*aa
    (ii) (aa)*(Ɛ + a) dan a*
    (iii) a (aa)*(Ɛ + a)b + b dan a*b

17. (i) a (ba + a)*b dan aa*b(aa*b)*
    (ii) A + a(a + b)* + (a + b)* aa(a + b)* dan ((b*a)*ab*)* Jelaskan (dalam frasa bahasa Inggris) bahasa yang terkait dengan yang berikut ini ekspresi reguler.
18. (i) (a + b)* a(A + bbbb)
    (ii) (a(a + bb)*)*

    (iii) (a(aa)*b(bb)*)*

    (iv) (b(bb)*)*(a(aa)*b(bb)*)*

    (v) (b(bb)*)*(a(aa)*b(bb)*)*(a(aa)*)*

    (vi) ((a + b)a)*
19. (D.N. Arden) Biarkan R, S, dan T menjadi tiga bahasa dan asumsikan itu. SEBUAH tidak ada di S. Buktikan pernyataan berikut.
    (i) Dari premis bahwa R = SR + T, kita dapat menyimpulkan bahwa R = S*T.
    (ii) Dari premis bahwa R = S*T, kita dapat menyimpulkan bahwa R = SR + T.
20. Jelaskan mengapa kita dapat mengambil pasangan ekspresi reguler yang setara dan ganti huruf a di keduanya dengan ekspresi reguler apa pun R dan hurufnya b dengan ekspresi reguler S apa pun dan ekspresi reguler yang dihasilkan akan memiliki bahasa yang sama. Misalnya, 15.(ii)

    $$(a*b)*a* = a*(ba*)*$$

    menjadi identitas

    $$(R*S)*R* = R*(SR*)*$$

    yang benar untuk semua ekspresi reguler R dan S. Secara khusus R = a + bb, S = ba* menghasilkan identitas yang rumit

    $$((a + bb)*(ba*))*(a + bb)* = (a + bb)* ((ba*)(a + bb)*)*$$

    Apa makna terdalam dari transformasi ini? Identitas apa yang akan dihasilkan dari penggunaan

    $$R = (ba*)* S = (A + b).$$

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

Bab 4

Deterministik Automata (FA)

4.1 Deterministik Automata (FA)

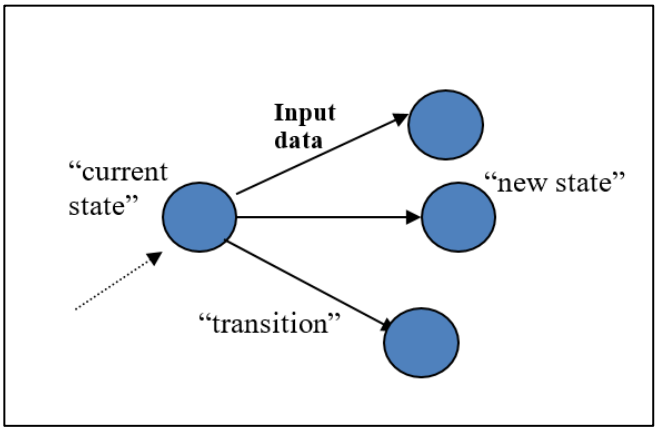
Kita pertama menemukan mesin automata yang sederhana, Finite automata sebagai mesin acceptor atau penerima bahasa yang berhingga karena hanya memiliki himpunan state berhingga. Disebut penerima bahasa karena memproses kata atau string dan menerima atau menolaknya, jadi kita bisa menganggapnya sederhana mekanisme pengenalan pola. Kita mulai dengan finite automata deterministik, atau DFA. kata sifat " deterministik " menandakan bahwa mesin automata memiliki satu dan hanya satu pilihan. Kami menggunakan DFA untuk mendefinisikan jenis bahasa tertentu, disebut bahasa biasa, dan dengan demikian buat hubungan pertama antara automata dan bahasa, tema yang sering diungkapkan dalam diskusi selanjutnya.

Definisi Finite Automata. Sebuah Mesin Finite automata didefinisikan sebagai kumpulan dari tiga hal:

a. Himpunan state berhingga, salah satunya ditetapkan sebagai keadaan awal, disebut state awal, dan beberapa (mungkin tidak ada) yang ditetapkan sebagai state akhir.
b. Alfabet Σ dari kemungkinan sebagai huruf input, dari mana string terbentuk,yang harus dibaca satu huruf pada satu waktu.
c. Himpunan transisi berhingga yang menyatakan untuk setiap state dan untuk setiap huruf dari alfabet sebagai input bergerak untuk menuju state berikutnya.

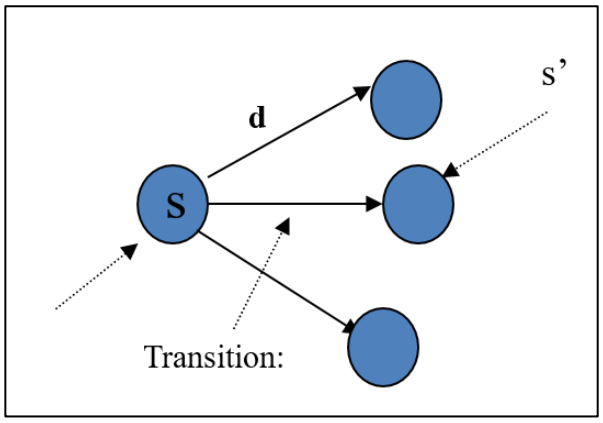
Misal String xxx tidak didalam bahasa L, yaitu semua string xxx yang dibaca oleh mesin FA dan tidak meraih state akhir, atau juga dapat dikatakan "xxx" adalah ditolak oleh FA ini." Jika tidak meraih pada state akhir sedangkan himpunan semua string yang meraih state akhir disebut bahasa yang didefinisikan diterima oleh mesin.

Pada gambar 3.1. ditunjukkan bagaimana mesin membaca input alphabet/kata dan kemudian menuju state berikutnya, menuju berikunya ini disebut dengan transisi (edge)



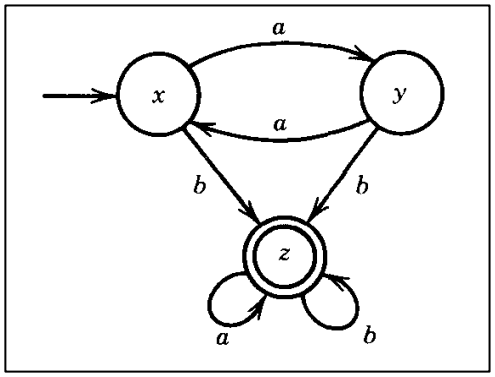
Gambar 4.1: State dengan input alphabet

Kami akan mendefinisikan automata yang menunjukkan state s dan beberapa data input d state yang akan dicapai

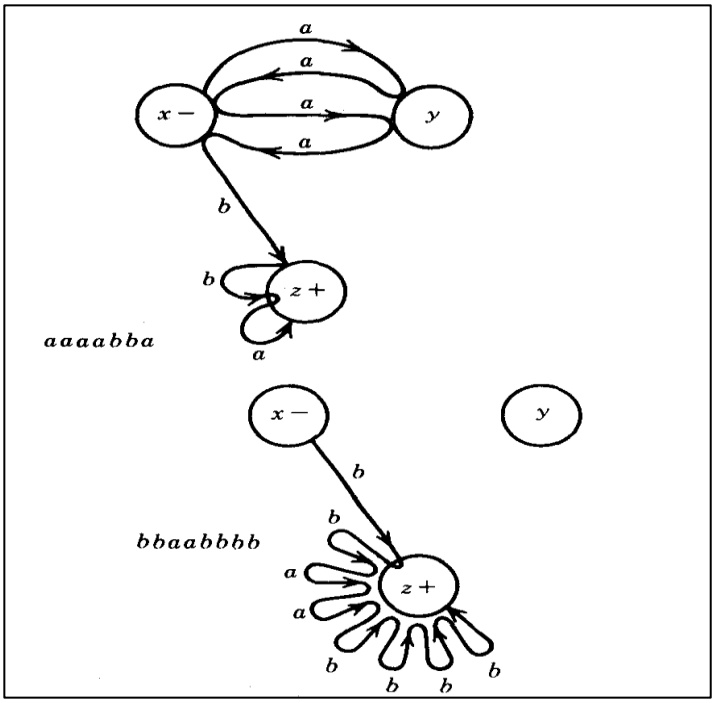


Gambar 4.2: Transisi state

Mari kita lihat gambar 3.3.a mesin FA membaca input string aaaabba dan bbaabbbb. Artinya bahwa proses mesin FA ini ketika menerima input string aaaabba, bbaabbbb mesin bergerak membaca string secara berurutan, sebagaimana gambar 3.3 (b). mesin FA, menerima aaaabba dan bbaabbbb.



Gambar 4.4: Finite automata membaca String/kata

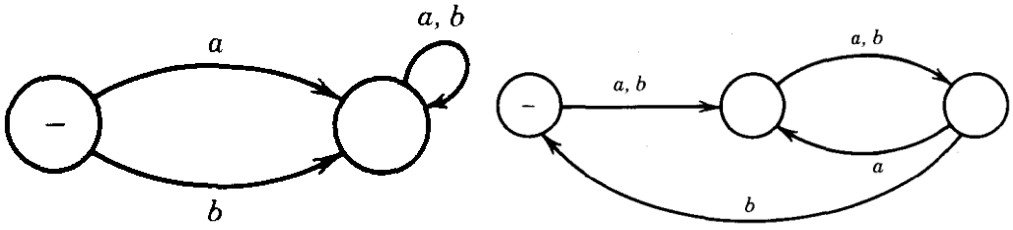


Gambar 4.5: Finite automata membaca String/kata

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

Gambar FA sebagai gambar lingkaran (State) dan grafik berarah disebut edge atau sisi/transisi, sedangkan lingkaran bertanda – atau awal panah adalah simbul state awal sedangkan lingkaran bertanda + atau double lingkaran disebut state akhir/penerimaan

Pada gambar 3.4. merupakan mesin automata yang tidak menerima kata aatau bahasapun karena mesin tidak memiliki state akhir, atau dengan kata lain jika mesin ini diberikan input kata apapun tidak dapat meraih final state



Gambar 4.5: Mesin FA tidak dapat menerima bahasa

4.2 Deterministik Finite Automata (DFA)

Sekarang saatnya untuk menyajikan tentang DFA yang menerima string/kata atau bahasa, sehingga mungkin mulai memperjelas beberapa argumen. Bahwa pada setiap input ada satu dan hanya satu state di mana mesin DFA dapat bertransisi dari state ke state lain.

Definisi :

Deteministik finite aotomata didefinisikan dengan;

$M = (Q,\Sigma, \delta, q0, F)$,

Q, adalah himpunan berhingga State

Σ adalah himpunan berhingga dari simbol-simbol yang disebut alfabet input

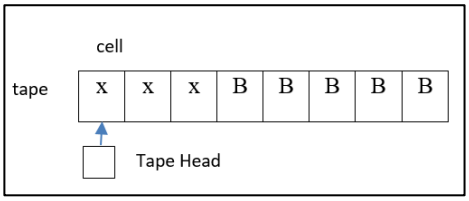
$\delta : Q \times \rightarrow Q$ adalah disebut fungsi transisi,

$q0 \in Q$ adalah State awal,

$F \in Q$ adalah himpunan State akhir.

Sebuah menin Abstrak DFA beroperasi dengan cara berikut.

Pada waktu awal, mesin automata berangkat dari state awal dengan menerima input simbul paling kiri dibaca oleh head masin simbul alphabet x.



Gambar 4.5: Tape dan Tape head

Ketika ujung tape tercapai, string atau kata diterima jika mesin berada di salah satu state penerimaan/final state. Jika tidak, string akan ditolak. Mekanisme input hanya dapat bergerak dari kiri ke kanan dan membaca tepat satu simbol pada setiap cell. Transisi dari satu state ke state yang lain diatur oleh fungsi transisi . Misalnya, jika

$$\delta (q0 , a) = q1,$$

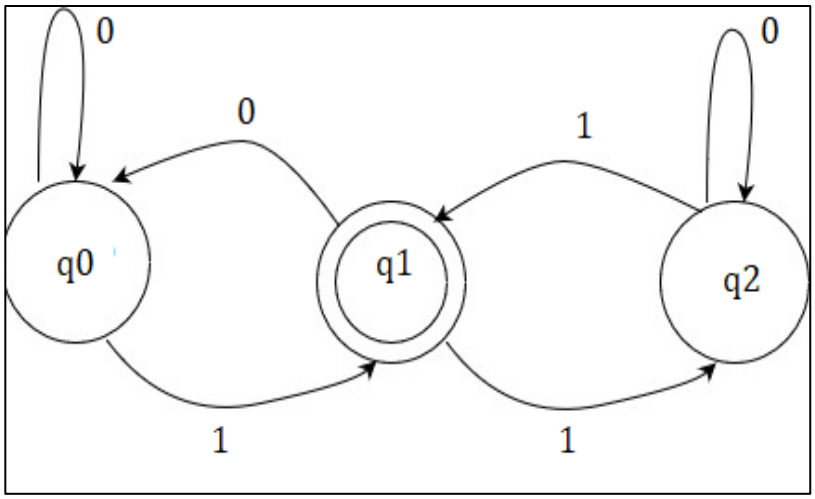
artinya bahwa dari state q0, dapat input alphabet a, kemudian menuju state berikutnya yaitu state q1.

Dalam membahas automata, sangat penting untuk memiliki pemahaman yang jelas yaitu dengan mengvisualisasikan finite automata, kami menggunakan graft transisi, di mana simpul-simpulnya mewakili state dan sisi-sisinya mewakili transisi. Label pada simpul adalah nama state, sedangkan label di sisi adalah simbol input. Misalnya, jika q0 dan q1 adalah state dari beberapa mesin DFAdi sebut dengan huruf M, maka M akan memiliki satu simpul berlabel q0 dan yang lain berlabel q1. Sisi (q0, q1) mewakili transisi (q0, a) = q1.

Secara formal dapat ditulis M = (Q,Σ, δ, q0, F), adalah deterministic finite automata sebagai penerima string/kata atau bahasa,

Contoh 1

M = ({q0, q1, q2} , {0, 1} , δ, q0, {q1})



Gambar 4.6: Deterministik finite automata

Dimana fungsi transisinya adalah,

δ (q0, 0) = q0, δ(q0, 1) = q1,

δ (q1, 0) = q0, δ(q1, 1) = q2,

δ (q2, 0) = q2, δ(q2, 1) = q1.

DFA ini menerima string 01. Mulai dari state q0, simbol 0 dibaca pertama. Melihat graph sisi, kita melihat bahwa mesin tetap di dalam state q0. Selanjutnya, mesin baca alphabet 1 mesin berada di state q1. dalam state q1 (state penerimaan). Oleh karena itu, maka string 01 diterima. DFA tidak menerima string 00, karena setelah membaca yang kedua 0, state berada dalam di q0. Dengan alasan yang sama, kita lihat bahwa mesin akan menerima string 101, 0111, dan 11001, tetapi tidak 100 atau 1100.

Bagaimana DFA proses membaca String, Hal pertama yang perlu kita pahami tentang DFA adalah bagaimana DFA "menerima" urutan simbol input. Atau "Bahasa" dari himpunan semua string yang diterima DFA. Misalkan 11001 adalah urutan simbol masukan. DFA bergerak mulai state awal melalui fungsi transisi sampai meraih urutan simbul terakhir yang berhenti di state penerimaan yaitu state q1

Contoh 2.1: secara formal DFA yang menerima semua barisan tring atau bahasa dan

dapat dinulis dengan, Bahasa L sebagai,

{w |w adalah dari bentuk x01y untuk semua string x dan y terdiri dari factor 0 dan 1}, sehingga L itu adalah

L= {1, 01,101,001,0111,11001,010101,……..,……..,……}

Tabel Transisi

Diagram transisi yang merupakan grafik automata seperti yang kita lihat di Bagian 2.1. kemudian ditabulasi bentuk,table transisi yang merupakan daftar tabular dari fungsi himpunan state input alphabet serta perubahan transisi transisinya, pada gambar 2.1. dapat ditabulasikan dalam bentuk

Tabel 4.1: Tabel Transisi

State	Input alphabet	
	0	1
Q0	Q0	Q1
Q1	Q0	Q2
Q2	Q2	Q1

Contoh gambar 3.7, merupakan transisi diagram DFA dengan M2 = (Q,Σ, δ, q0, F),

Contoh 2

Ditunjukan pada diagram mesin DFA sebagaimana berikut dimana secara formal dari mesin DFA adalah

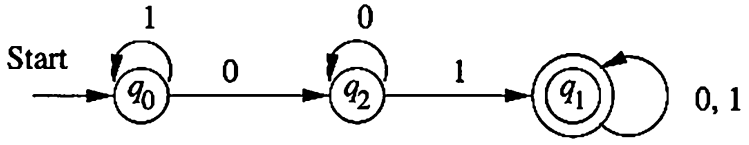
M3 = (Q,Σ, δ, q0, F),

Dimana Q = { q0, q1, q2,}

Σ = {0,1}

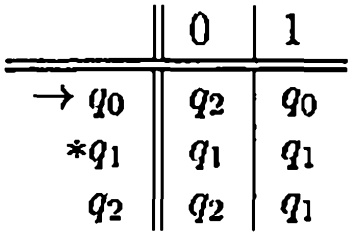
S ={q0}

F ={ q1}



Gambar 4.7: Diagram Mesin DFA

Dari diagram ini dapat di rubah atau di konversi menjadi table transisi, sebagaimana berikut,



Gambar 4.8: Table Transisisi DFA

Dimana pada gambar diatas dari q0, dapat input alphabet 0, menuju state q2, kita masih berada di q0, q0 dapat input alphabet 1, menuju ke state q0, selanjutnya kita berada di state q1, dapat input alphabet 0, kita kembali ke q1, demikian pula q1 dapat input alphabet 1, kita juga kembali ke state 1, sekarang kita berada di state q2, q2 dapat input alphabet 0, kita kembali ke state q2, kemudian kita masih di state q2, dapat input alphabet 1, kita berhenti di state 1., sehingga menjadi gambar 2.5

Contoh 3.

Ditunjukkan diagram transisi sebagaimana gambar 4, dimana Mesin DFA didefinisikan dengan,

M4 = (Q,Σ, δ, q0, F), Q = { q0, q1, q2, q3 }, Σ = {0,1}, S ={q0}, F ={ q0},

δnya

old stete input Σ

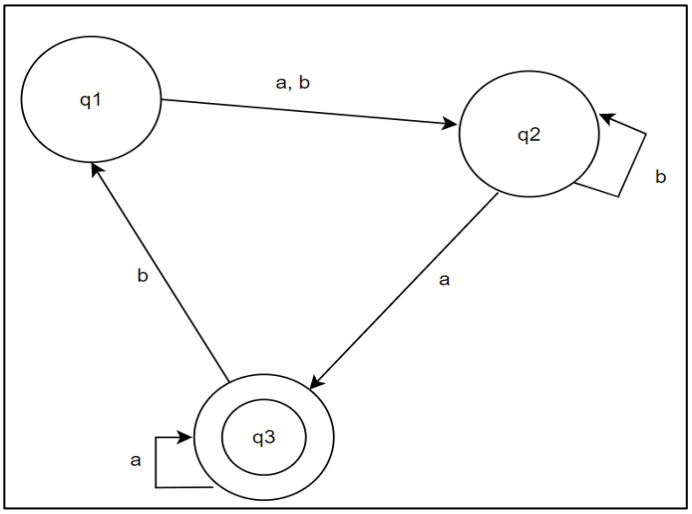
	0	*1*
'q0	*'q3*	*'q1*
'q1	*'q3*	*'q0*
'q2	*'q0*	*'q3*
'q3	*'q1*	*'q32*

Contoh disertakan dengan bahasa pemrograman

Contoh 4

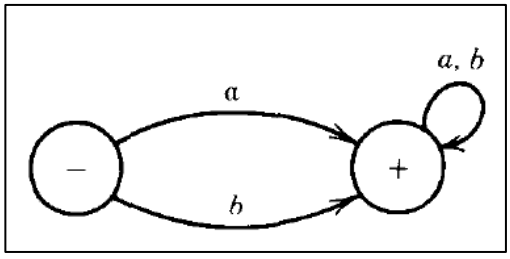
Ditunjukkan diagram transisi sebagaimana gambar 4, dimana Mesin DFA didefinisikan dengan,

M4 = (Q,Σ, δ, q0, F), Q = { q1, q2, q3 }, Σ = {a,b}, S ={q1}, F ={ q3},



Gambar 4.9: Table transisisi DFA

CONTOH DFA



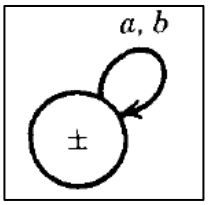
Gambar 4.10: DFA

Pada contoh yang lain dijelaskan bahwa a, b sebuah Pada gambar di atas kita telah menggambar satu sisi dari state di bagian kanan belakang ke dalam dirinya sendiri dan diberi loop ini dua label a dan b, dipisahkan oleh koma artinya ini adalah jalan yang dilalui jika salah satu huruf dibaca. (Kami menyelamatkan diri kami sendiri dari menggambar tepi loop kedua.) Sepintas sepertinya mesin ini menerima segalanya. Huruf pertama dari input membawa kita ke state sebelah kanan dan, sekali di sana, kita terjebak selamanya. Ketika string input habis, ada kita berada dalam kondisi akhir yang benar. Deskripsi ini, bagaimanapun, menghilangkan kemungkinan bahwa inputnya adalah string null Ɛ. Jika string input adalah string null, kita ditinggalkan kembali di state bagian kiri, dan kami tidak pernah sampai ke state akhir. Di sana adalah masalah kecil tentang memahami bagaimana mungkin Ɛ pernah menjadi string masukan ke sebuah. FA, karena string, menurut definisi, dieksekusi (dijalankan) oleh membaca surat-suratnya satu per satu. Dengan konvensi kita akan mengatakan bahwa Ɛ dimulai di state awal dan kemudian berakhir di sana di semua FA. Bahasa yang diterima oleh mesin ini adalah himpunan semua string kecuali Ɛ. Sehingga definisi ekspresi reguler sebagaimana berikut

$$(a + b)(a + b)^* = (a + b)+$$

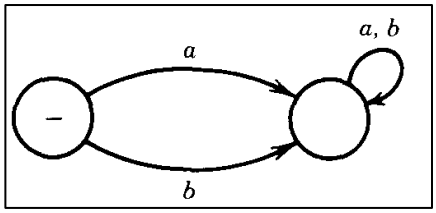
CONTOH

Salah satu dari banyak FA yang menerima semua kata adalah:

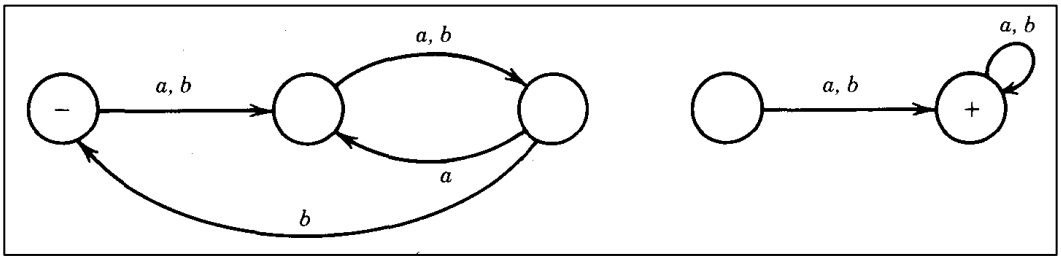


Di sini tanda ± berarti bahwa keadaan yang sama adalah state awal dan state akhir.

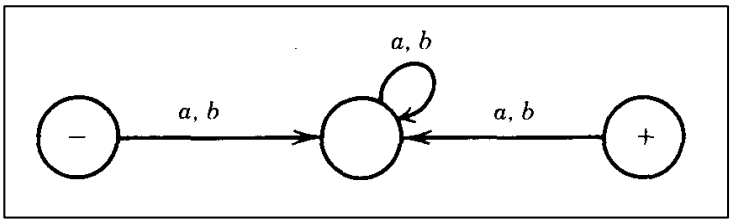
Karena hanya ada satu state dan apa pun yang terjadi kita harus tetap di sana, bahasa untuk mesin ini adalah: (a + b)* Demikian pula, ada FA yang tidak menerima bahasa. Ini adalah dua jenis: FA yang tidak memiliki status akhir, seperti



dan FA di mana lingkaran yang mewakili state akhir tidak dapat dicapai dari state awal. Ini mungkin karena gambar di dua terpisah seperti



(dalam hal ini kami mengatakan bahwa gambar terputus) atau karena alasan seperti yang ditunjukkan di bawah ini.

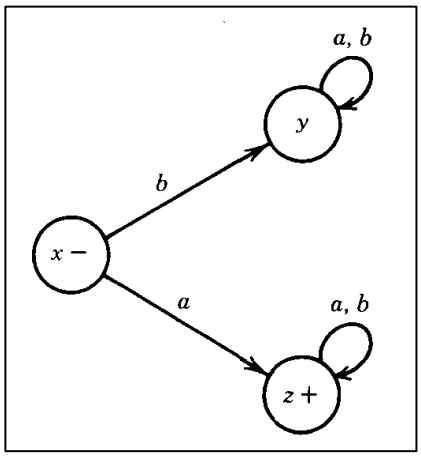


CONTOH

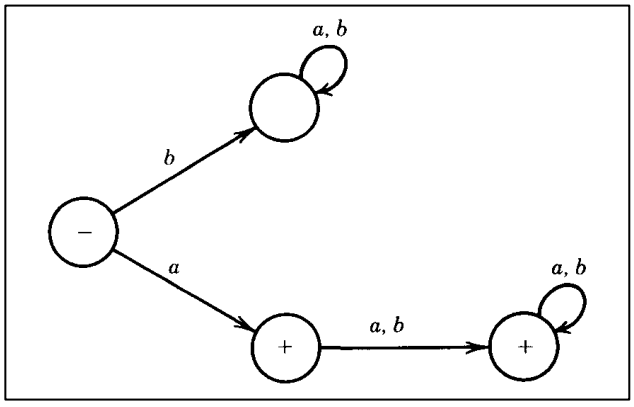
Misalkan kita ingin membuat otomat terbatas yang menerima semua kata dalam Bahasa

$$a(a + b)^*$$

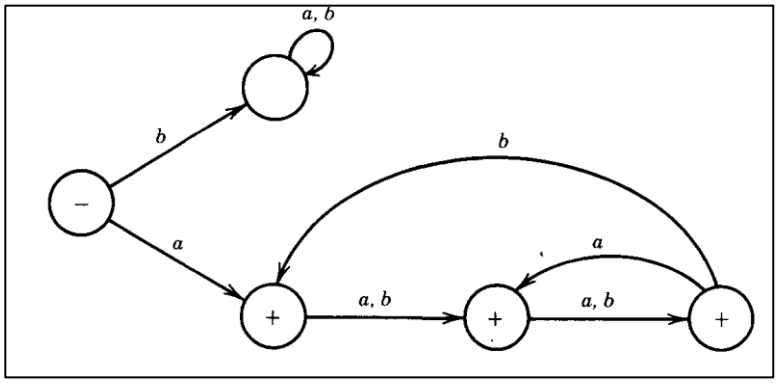
yaitu semua string yang dimulai dengan huruf a. Kita mulai dari state x dan, jika huruf pertama yang dibaca adalah a b kita pergi ke state buntu y. (Sebuah "keadaan buntu" adalah cara informal untuk menggambarkan state yang tidak dapat ditinggalkan oleh string setelah itu inputkan.) Jika huruf pertama adalah a, kita pergi ke state buntu z, di mana z adalah state akhir. Mesinnya terlihat seperti ini:



Bahasa yang sama dapat diterima oleh mesin empat state, seperti di bawah ini:



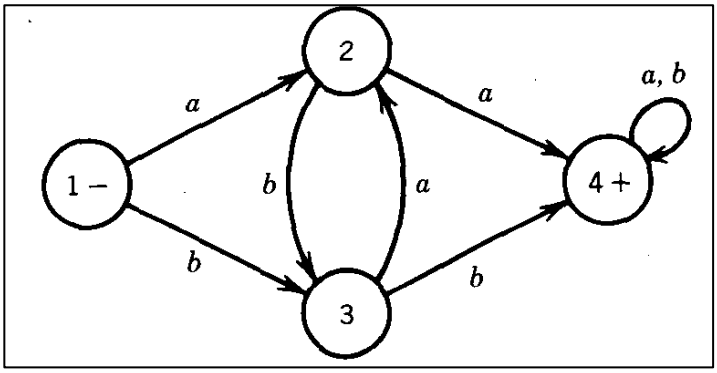
Hanya kata a yang diakhiri dengan state + pertama. Semua kata lain yang dimulai dengan an mencapai dan menyelesaikan dalam state + kedua di mana mereka diterima. Hal ini dapat dibawa lebih jauh ke FA lima state a bagian seperti di bawah ini:



Contoh di atas adalah FA yang memiliki lebih dari satu state akhir. Dari mereka kita dapat melihat bahwa tidak ada mesin yang unik untuk bahasa tertentu. Kami kemudian dapat mengajukan pertanyaan, "Apakah selalu ada setidaknya satu FA yang menerima setiap bahasa yang mungkin? Lebih tepatnya, jika L adalah beberapa bahasa, apakah pasti ada mesin jenis ini yang menerima input dalam L secara tepat, sementara menolak semua yang lain?" Kita akan segera melihat bahwa pertanyaan ini terkait dengan pertanyaan, "Bisakah semua bahasa diwakili oleh ekspresi reguler?" Kami membuktikan, disebelumnya, bahwa setiap bahasa yang dapat diterima oleh FA dapat didefinisikan oleh ekspresi reguler dan, sebaliknya, setiap bahasa yang dapat didefinisikan oleh ekspresi reguler dapat diterima oleh beberapa FA. Namun, kami akan melihat bahwa ada bahasa yang tidak dapat didefinisikan oleh ekspresi reguler juga tidak diterima oleh FA. Ingat, agar bahasa menjadi bahasa diterima oleh FA tidak hanya berarti bahwa semua kata dalam bahasa tersebut berjalan ke state akhir tetapi juga tidak ada string yang tidak ada dalam bahasa tersebut. Mari kita pertimbangkan beberapa contoh lagi dari FA.

CONTOH

Pertimbangkan FA yang digambarkan di bawah ini:



Sebelum kita mulai memeriksa bahasa apa yang diterima mesin ini, mari kita melacak jalur yang terkait dengan beberapa string input tertentu. Mari kita inputkan abab. Kita mulai dari state awal 1. Huruf pertama adalah a, jadi dibutuhkan kita untuk menyatakan 2. Dari sana huruf berikutnya, b, membawa kita ke state 3. Selanjutnya huruf, a, kemudian membawa kita kembali ke state 2. Huruf keempat adalah a b dan itu mengambil kita untuk menyatakan 3 lagi. Huruf terakhir adalah a yang mengembalikan kita ke state 2 di mana kita berakhir. State 2 bukanlah state akhir (tidak ada +), jadi kata ini tidak diterima.

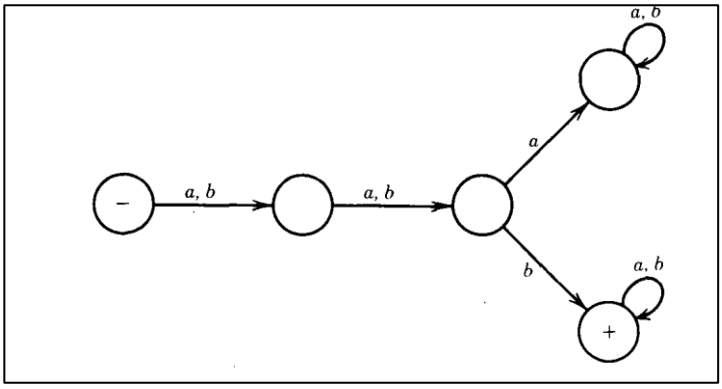
Mari kita telusuri kata babbb. Seperti biasa, kita mulai di state 1. Huruf pertama b membawa kita ke state 3. Ɛ kemudian membawa kita ke state 2. Huruf ketiga b mengambil kita kembali ke keadaan 3. Sekarang b lain membawa kita ke keadaan 4. Setelah di state 4, kita tidak bisa keluar tidak peduli sisa string. Setelah di state 4 kita harus tetap dalam state 4, dan karena itu adalah state terakhir, string diterima. Ada dua cara untuk mencapai status 4 di FA ini. Salah satunya dari negara state 2, dan yang lain dari state bagian 3. Satu-satunya cara untuk sampai ke state 2 adalah dengan membaca input huruf a (baik saat .dalam state 1 atau dalam state 3). Jadi ketika kita dalam state 2 kita tahu kita baru saja. membaca sebuah. Jika kita langsung membaca a yang lain, kita langsung menuju ke state 4. Demikian pula dengan keadaan 3. Untuk sampai ke keadaan 3 kita perlu membaca a.b. Setelah

di state 3, jika kita langsung membaca b yang lain, kita menuju ke state 4; jika tidak, kita pergi ke state 2. Setiap kali kita menemukan substring aa dalam string input,. pertama membawa kita ke state 4 atau state 2. Bagaimanapun, a berikutnya membawa kita ke state 4. The situasi dengan bb.Singkatnya, kata-kata yang diterima oleh mesin ini persis seperti string itu yang memiliki huruf ganda di dalamnya. Bahasa ini, seperti yang telah kita lihat, juga dapat didefinisikan oleh ekspresi reguler sebagai berikut

$$(a + b)^*(aa + bb)(a + b)^*$$

CONTOH

Mari kita pertimbangkan FA yang digambarkan di bawah ini:

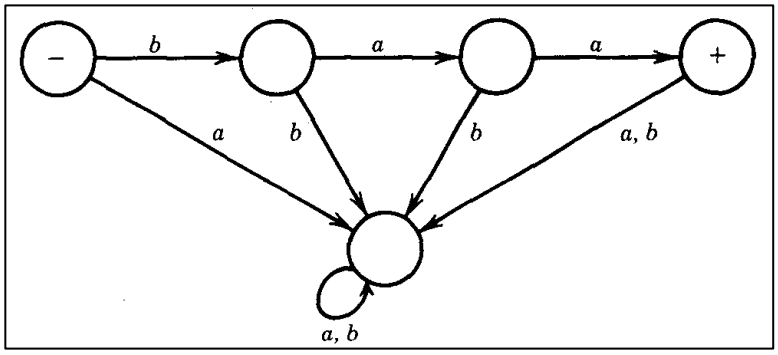


Mesin ini akan menerima semua kata dengan b sebagai huruf ketiga dan menolak semua kata lain. Beberapa state bagian pertama hanya menunggu state yang memaca urutan dua huruf pertama input. Kemudian datang sate penerimaan. Sebuah kata yang memiliki lebih sedikit dari tiga huruf tidak dapat memenuhi syarat, dan jalurnya berakhir di salah satu dari kiri atas menyatakan, tidak ada +. Beberapa ekspresi reguler yang mendefinisikan set ini adalah

$$(aab + abb + bab + bbb)(a + b)^* \text{ dan } (a + b)(a + b)(b)(a + b)^* = (a + b)2b\ (a + b)^*$$

CONTOH

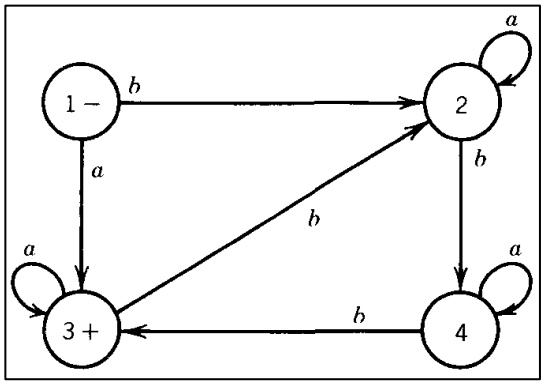
Mari kita pertimbangkan FA yang sangat khusus, yang hanya menerima kata baa.



Bahasa yang diterima oleh FA ini adalah L {baa} saja

CONTOH

Mari kita ambil contoh yang lebih rumit. Pertimbangkan FA yang ditunjukkan di bawah ini:



Bahasa apa yang diterima oleh mesin ini? Kita mulai dari state 1, dan jika kita membaca kata yang dimulai dengan a, kita langsung menuju ke state akhir 3. Kita dapat tetap berada di state 3 selama kita terus membaca hanya a. Karena itu, semua kata dalam bentuk

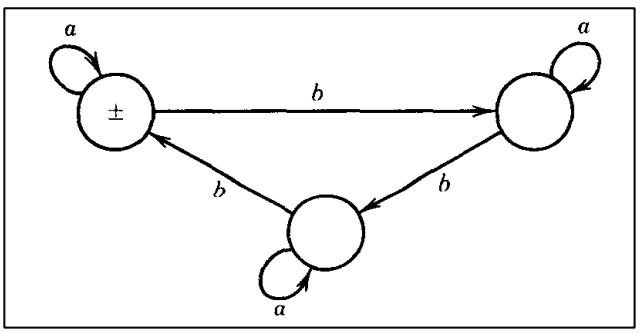
aa*

untuk menyatakan 3 tetapi kemudian kita membaca a b? Ini kemudian membawa kita ke state 2. Untuk mendapatkan kembali ke state akhir, kita harus melanjutkan ke state 4 dan kemudian ke state 3. Ini perjalanan membutuhkan dua b lagi untuk dibaca sebagai input. Perhatikan bahwa di state 2, 3, dan 4 semua a yang dibaca diabaikan. Hanya b yang menyebabkan perubahan state. Rekapitulasi apa yang kita ketahui: Jika string input dimulai dengan a dan kemudian memiliki beberapa b, harus memiliki tiga b untuk mengembalikan kita ke keadaan 3, atau enam b untuk melakukan perjalanan (state 2, state 4, dan state 3) dua kali, atau 9 b atau 12 b . ... Di dengan kata lain, string input dimulai dengan a dan memiliki jumlah total b habis dibagi 3 diterima. Jika dimulai dengan a dan memiliki total jumlah b tidak habis dibagi 3, maka inputnya ditolak karena jalurnya melalui mesin berakhir pada keadaan 2 atau keadaan 4. Apa yang terjadi pada string input yang dimulai dengan b? Ia menemukan dirinya dalam state 2 dan membutuhkan dua b lagi untuk mencapai state 3 (b ini dapat dipisahkan oleh sejumlah a).

Setelah di state bagian 3, tidak perlu lagi b, atau tiga lagi b, atau enam b lagi, dan seterusnya. Semua dan semua, string input, baik yang dimulai dengan a atau b harus memiliki jumlah total b yang habis dibagi 3 diterima. Juga jelas bahwa setiap string yang memenuhi persyaratan ini akan mencapai state akhir. Bahasa yang diterima oleh mesin ini dapat didefinisikan dengan ekspresi reguler

a*(a*ba*ba*ba*)*(a + a*ba*ba*ba*)

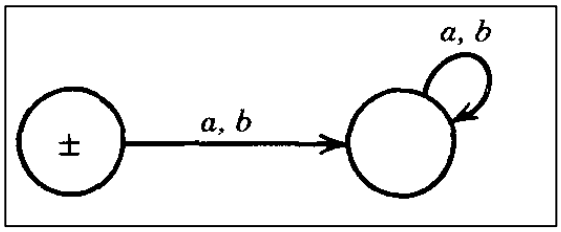
Satu-satunya tujuan untuk faktor terakhir adalah untuk menjamin bahwa Ɛ bukanlah suatu kemungkinan karena tidak diterima oleh mesin. Jika kami tidak keberatan Ɛ dimasukkan dalam bahasa tersebut, kami dapat menggunakan FA yang lebih sederhana ini.



Ekspresi reguler (a + ba*ba*b)+ juga mendefinisikan bahasa asli (non-Ɛ).

CONTOH

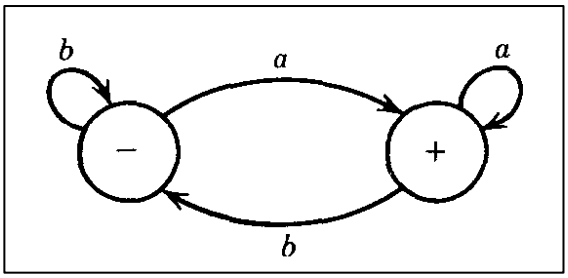
FA berikut hanya menerima kata Ɛ



Perhatikan bahwa state kiri adalah state awal dan state akhir. Semua kata lain dari Ɛ pergi ke state yang tepat dan tinggal di sana.

Contoh

Perhatikan FA berikut ini:

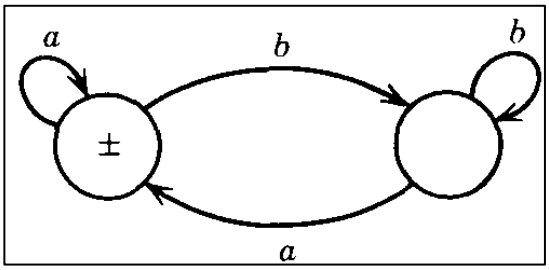


Tidak peduli di State mana kita berada, ketika kita membaca a, kita pergi ke tangan kanan state dan ketika dibaca a b kita pergi ke state sebelah kiri. Masukan apa saja string yang diakhiri dengan State + harus diakhiri dengan huruf a, dan string apa pun yang diakhiri dalam a harus diakhiri dengan +. Oleh karena itu, bahasa yang diterima oleh mesin ini adalah

$$(a + b)^*a$$

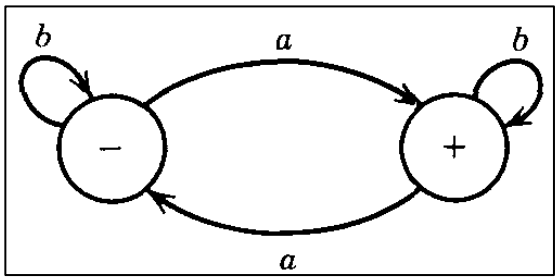
Contoh

Bahasa dalam contoh di atas tidak termasuk Ɛ. Jika kita menambahkan Ɛ, kita mendapatkan bahasa semua kata yang tidak berakhiran b. Ini diterima oleh FA di bawah



Contoh

Perhatikan FA berikut ini:



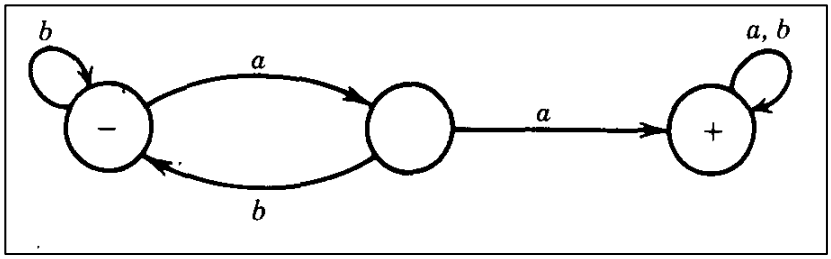
Satu-satunya huruf yang menyebabkan gerakan antar keadaan adalah a, beberapa b mesin dalam state yang sama. Kita mulai dari -. Jika kita membaca a pertama, kita pergi ke +. Detik a membawa kita kembali. Sepertiga a

membawa kita ke + lagi. Kami berakhir di + setelah yang pertama, ketiga, kelima, ketujuh, . . . sebuah. Bahasa yang diterima oleh ini mesin adalah semua kata dengan bilangan ganjil a.

b*a(b*ab*ab*)*

Contoh

Perhatikan FA berikut ini:

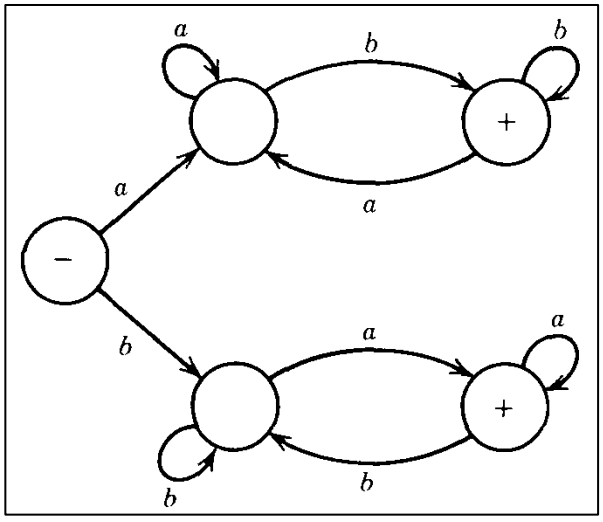


Mesin ini akan menerima bahasa semua kata a ganda mereka di suatu tempat. Kami tetap dalam keadaan awal sampai kami membaca a. Ini membawa kita ke State tengah. Jika huruf berikutnya adalah a lain, kita pindah ke State + di mana kita harus tinggal dan diterima. Jika huruf berikutnya adalah b, bagaimanapun, kita kembali ke - untuk menunggu a berikutnya. Sebuah a diikuti oleh a b akan membawa kita dari - ke tengah ke -, sementara a diikuti oleh wasiat membawa kita dari - ke tengah ke +. Bahasa yang diterima oleh mesin ini juga dapat ditentukan oleh regular ekspresi

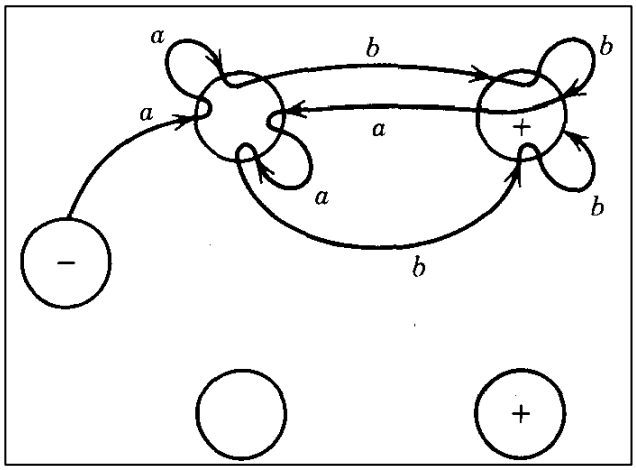
(a + b)*aa(a + b)*

Contoh

FA berikut menerima semua kata yang memiliki huruf pertama dan terakhir yang berbeda. Jika kata dimulai dengan a, untuk diterima harus diakhiri dengan b dan sebaliknya

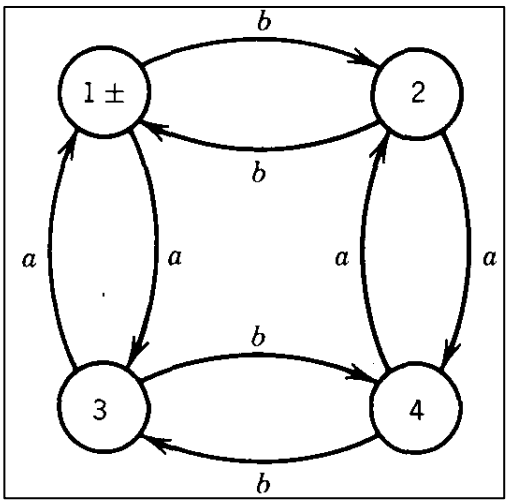


Jika kita mulai dengan a, kita mengambil jalan dan melompat bolak-balik antara dua state atas yang berakhir di sebelah kanan (di +) hanya jika huruf terakhir membaca adalah b. Jika huruf pertama yang dibaca adalah b, kita sampai ke + di bagian bawah hanya ketika kita membaca a sebagai huruf terakhir. Ini dapat dipahami lebih baik dengan memeriksa jalur melalui FA dari string input aabbaabb, seperti yang ditunjukkan di bawah ini:



Contoh

Sebagai contoh terakhir dari FA dalam bab ini, mari kita perhatikan gambar di bawah ini:



Untuk memproses string huruf, kita mulai dari state 1, yang ada di kiri atas dari gambar. Setiap kali kita menemukan huruf a dalam string input yang kita ambil sebuah kereta api. Ada empat sisi yang diberi label a. Semua tepinya juga menandai jalan dari salah satu dari dua state atas (state 1 dan state 2) ke salah satu dari yang lebih rendah dua state (state 3 dan state 4) atau dari salah satu dari dua state yang lebih rendah ke salah satu dari dua state atas. Jika kita ke utara dan kita membaca a, kita pergi ke selatan. Jika kita ke selatan dan kita membaca a, kita pergi ke utara. Huruf a membalikkan state naik/turun. Apa yang terjadi pada kata yang diterima dan berakhir kembali di state 1? Tanpa mengetahui hal lain tentang string, kita dapat mengatakan bahwa itu pasti ada memiliki jumlah genap di dalamnya. Setiap a yang membawa kita ke selatan seimbang oleh beberapa a yang membawa kami kembali ke utara. Kami melewati garis Mason-Dixon secara seimbang beberapa kali, satu untuk setiap a. Jadi setiap kata dalam bahasa FA ini memiliki jumlah genap di dalamnya. Juga, kita dapat mengatakan bahwa setiap string input dengan jumlah a yang genap akan menyelesaikan lintasannya di utara (negara bagian 1 atau state 2). Ada lagi yang bisa kami katakan tentang kata-kata yang diterima oleh ini mesin. Ada empat sisi berlabel b. Setiap sisi berlabel b mengambil kami dari salah satu dari dua state di sebelah kiri gambar (state 1 dan state 3) ke salah satu dari dua state di sebelah kanan (state 2 dan state 4) atau yang lain membawa kita dari salah satu dari dua negara di sebelah kanan ke salah satu dari dua negara di sebelah kiri.

Setiap b yang kita temui di input adalah pembalik timur/barat. Jika kata dimulai keluar di state 1, yang ada di sebelah kiri, dan berakhir kembali di state 1 (di sebelah kiri), itu pasti telah melintasi beberapa kali. Oleh karena itu, semua kata-kata dalam bahasa yang diterima oleh FA ini memiliki jumlah b yang genap serta bilangan genap dari a. Kita juga dapat mengatakan bahwa setiap string input dengan jumlah b yang genap akan meninggalkan kita di barat (state 1 atau state 3). Ini adalah satu-satunya dua kondisi pada bahasa. Semua kata dengan genap jumlah a dan bilangan genap b harus kembali ke state 1. Semua kata yang kembali ke state I. Semua kata yang

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

berakhiran state 2 telah melewati garis beberapa kali tetapi telah melewati beberapa kali ganjil; oleh karena itu mereka memiliki jumlah genap a dan bilangan ganjil dari b. Semua kata yang berakhiran state 3 memiliki bilangan b genap tetapi bilangan a ganjil. Semua kata yang berakhiran state 4 memiliki bilangan ganjil a dan bilangan ganjil b. Jadi sekali lagi kita lihat bahwa semua kata BAHKAN-GENAP harus diakhiri dengan state 1 dan diterima. Satu ekspresi reguler untuk bahasa BAHKAN-BAHKAN dibahas di rinci pada bab sebelumnya. Perhatikan betapa 'lebih mudah untuk memahami FA daripada ekspresi reguler. Kedua metode pendefinisian bahasa memiliki kelebihan, tergantung pada aplikasi yang diinginkan. Tapi kami masih jauh dari mempertimbangkan aplikasi.

Dengan bahasa Bahasa pemrograman Golang miliknya google

Source Code

```
package main
import (
	"fmt"
)
type State struct {
	nama_state string
	final     bool
	a         *State
	b         *State
}
var q1 State
var q2 State
var q3 State
func initState() {
	q3.nama_state = "q3"
	q3.final = true
	q3.a = &q3
	q3.b = &q1
	q2.nama_state = "q2"
	q2.final = false
	q2.a = &q3
	q2.b = &q2
	q1.nama_state = "q1"
	q1.final = false
	q1.a = &q2
	q1.b = &q2
}
func info() {
	fmt.Println("Nama Kelompok: ")
```

```
        fmt.Println("1. Reyhan Haqiqi Alif F.\t(201080200044)")
        fmt.Println("2. M. Fadli Zaka\t\t(201080200007)")
        fmt.Println("3. Imron Raafisianto\t\t(201080200087)")
        fmt.Println("\nMESIN PENERIMA BAHASA")
        fmt.Println("================================")
        fmt.Println("State awal = q1")
        fmt.Println("State akhir = q3")
        fmt.Println("Input = a | b")
}
func utama() {
        info()
        var bahasa []byte
        initState()
        var currState *State = &q1
        fmt.Print("\nMasukkan Bahasa : ")
        fmt.Scanln(&bahasa)
        fmt.Printf("L(G)={%s}\n", bahasa)
        for _, v := range bahasa {
                switch string(v) {
                case "a":
                        fmt.Printf("%s => a = %s\n", currState.nama_state, currState.a.nama_state)
                        currState = currState.a
                case "b":
                        fmt.Printf("%s => b = %s\n", currState.nama_state, currState.b.nama_state)
                        currState = currState.b
                default:
                        fmt.Printf("%s tidak dimengerti\n", string(v))
                  }
}
if currState.final {
          fmt.Print("Bahasa diterima oleh mesin\n\n")
} else {
          fmt.Print("Bahasa tidak diterima oleh mesin\n\n")
      }
}

func main() {
```

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

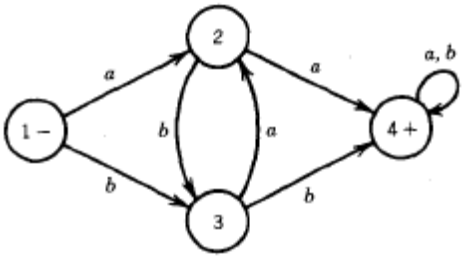
```
        lanjut := true
        var pilih string
        for lanjut {
          utama()
          fmt.Print("Ingin mencoba program lagi ? (Y/N) ")
          fmt.Scanln(&pilih)
          if pilih == "Y" || pilih == "y" {
                    lanjut = true
           } else {
                    lanjut = false
                    }
         }
}
```

Mesin DFA tersebut menghasilkan bahasa yang diterima sebagai berikut,

Language L ={ aa, ba,aba.bba,abba,aaaa,baaa, baabaa,bbbaabab,…….,……..,……..}.

4.6 Latihan Soal

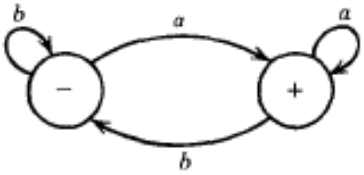
1. Ditunjukkan gambar DFA sebagai berikut



M4 = (Q,Σ, δ, q0, F), Q = { 1,2,3,4 }, Σ = {a,b}, S ={1}, F ={ 4},

   a. Buktikan bahwa string-string abaa, babb, aaab,bbbba. Diterima oleh mesin DFA
   b. Buatkan table transisinya

2. Ditunjukkan gambar DFA sebagai berikut



   a. Buktikan bahwa string-string abaa, babb, aaab,bbbba. Diterima oleh mesin DFA
   b. Buatkan table transisinya

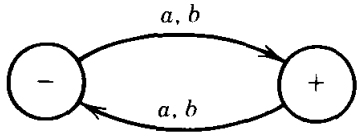
3. Ditunjukkan table transisi dari mesin DFA

		a	*b*
start	*x*	*y*	*z*
	y	*x*	*z*
final	*z*	*z*	*z*

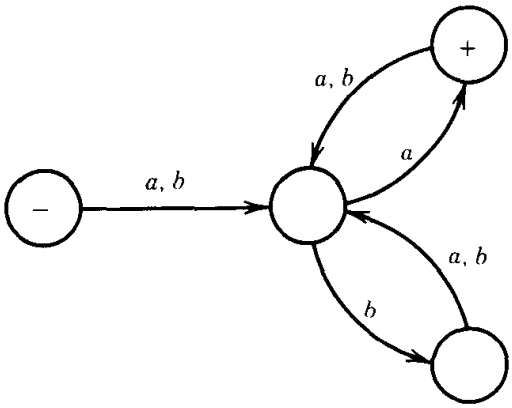
a. Tentukan diagram mesin DFA
b. Tentukan bahasa yang diterimanya, minimal 10 kata
c. Temukan dua FA yang memenuhi kondisi ini: Di antara mereka, mereka menerima semua kata dalam (a + b)*, tetapi tidak ada kata yang diterima oleh kedua mesin.

4. Jelaskan bahasa yang diterima oleh FA berikut.

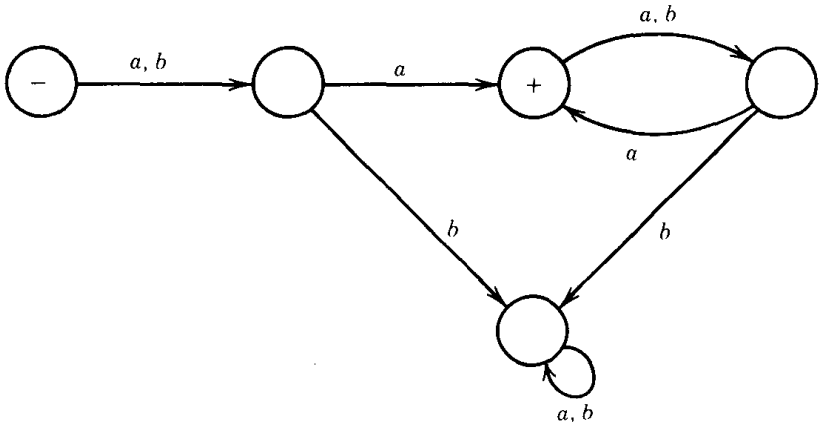
(i)



(ii)

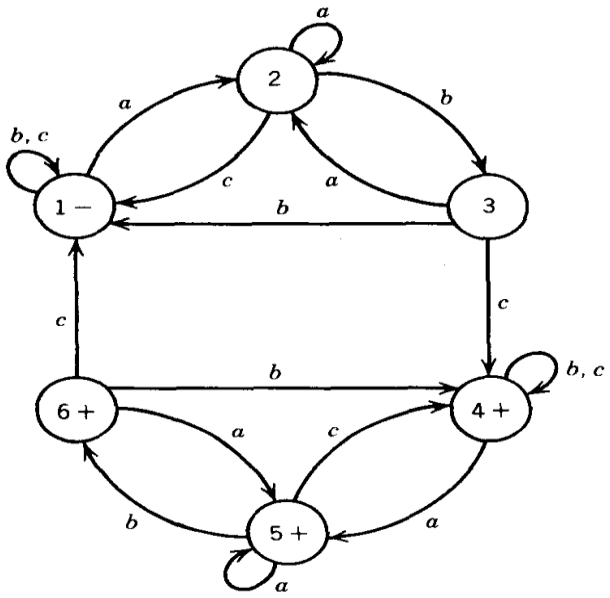


(iii)

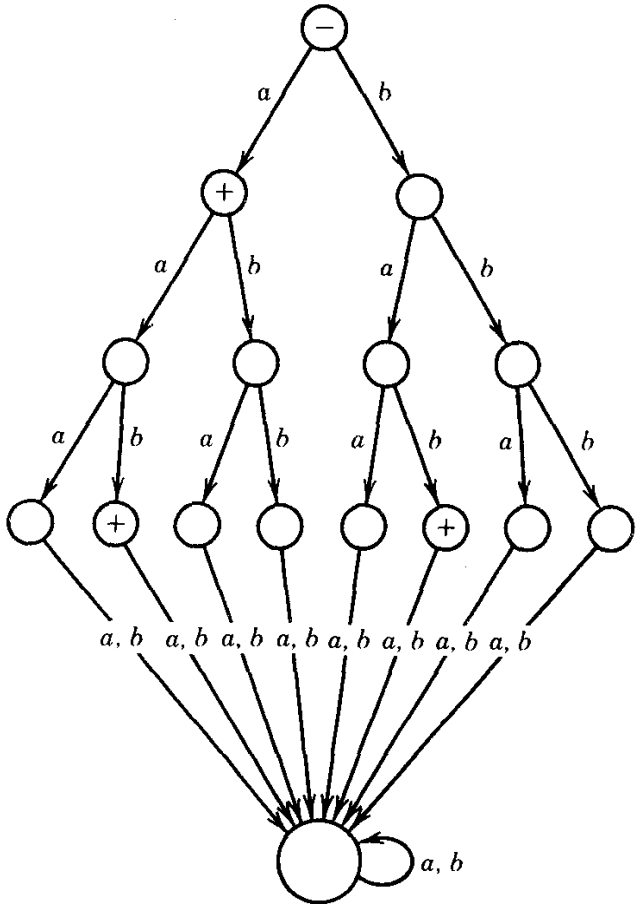


(iv). Tulis ekspresi reguler untuk bahasa yang diterima oleh ketiganya mesin.

5. Berikut ini adalah FA atas alfabet I = {a, b, c}. Buktikan itu menerima semua string yang memiliki jumlah kemunculan substring yang ganjil abc.



6. Perhatikan FA berikut ini:



(i) Tunjukkan bahwa string input apa pun dengan lebih dari tiga huruf tidak diterima oleh FA ini.
(ii) Tunjukkan bahwa satu-satunya kata yang diterima adalah a, aab, dan bab.

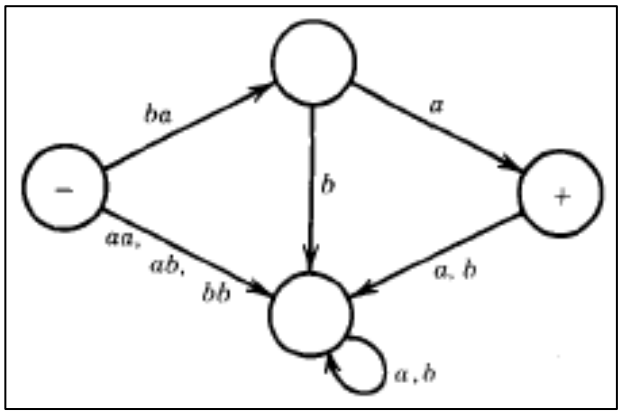
(iii) Tunjukkan bahwa dengan mengubah tanda + saja kita dapat membuat FA ini menerima bahasa {bb, aba, bba}
(iv) Tunjukkan bahwa bahasa apa pun yang kata-katanya kurang dari empat surat dapat diterima oleh mesin yang terlihat seperti ini dengan + tanda di tempat yang berbeda.
(v) Buktikan bahwa jika L adalah bahasa terhingga, maka ada beberapa FA yang menerima L

Bab 5

Transition Graph

5.1 Definisi Transition Graph

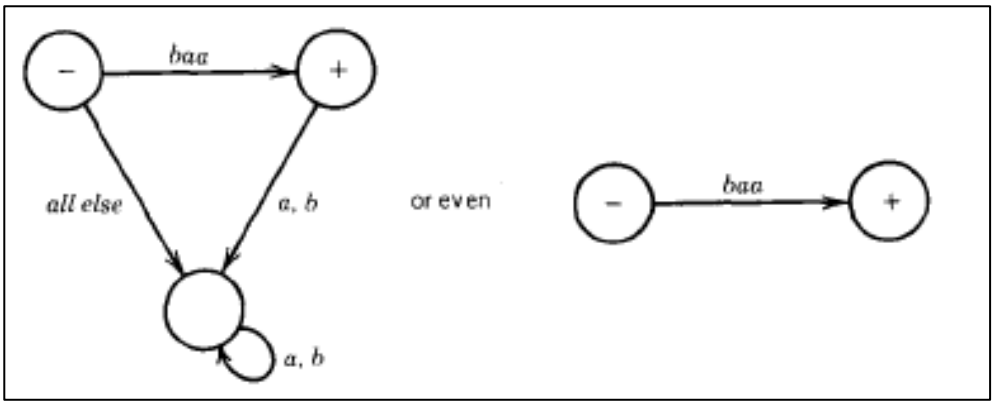
Pada bab sebelumnya kita sudah membahas tentang finite automta deterministic yang menerima himpunan string/kata atau bahasa, Contoh yang diberikan adalah mesin FA yang membaca satu alphabet dalam satu waktu dan menuju state berikutnya saja, sekarang kita merancang mesin yang lebih kuat yang bisa membaca satu atau dua huruf input string pada satu waktu dan dapat mengubah state menuju state berikutnya, sebagaimana mungkin desain mesin seperti di bawah ini:



Gambar 5.1: Transition Graph

Objek yang kita bahas dalam bab ini adalah model matematika, yang akan kita temukan adalah abstraksi dan penyederhanaan tentang bagaimana mesin bekerja. Aturan baru untuk model ini juga akan menjadi praktis. Ini akan memudahkan kita untuk merancang mesin yang menerima bahasa berbeda. Mesin di atas juga dapat membaca dari string input satu atau dua huruf sekaligus, tergantung state mana. Perhatikan bahwa di mesin ini sisi memiliki beberapa label yang dipisahkan dengan koma seperti gambar 4.1. menunjukkan bahwa jika huruf input adalah salah satu kombinasi yang ditunjukkan. Jika kita tertarik pada mesin yang hanya menerima kata baa, mengapa berhenti? dengan asumsi bahwa mesin hanya dapat membaca dua huruf sekaligus? Sebuah mesin yang menerima kata ini dan yang dapat membaca hingga tiga huruf sekaligus dari string input dapat dibangun dengan state yang di sederhanakan(digabung.

Pada gambar 5.2, FA didefinisikan mesin penerima kata adalah dari state awal hanya menuju state akhir saja, yaitu ketika mesin bekerja membaca string baa hingga meraih state akhir, sedangkan dari state awal menuju state lainnya, tidak didefinisikan sebagai mesin yang bekerja, karena mereka terjerebak ke state bukan state akhir/penerima

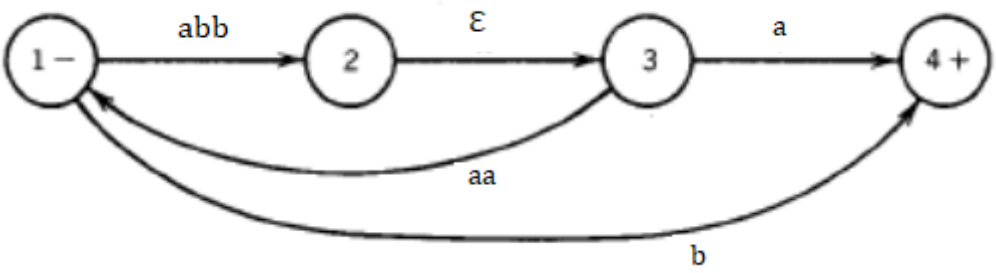


Gambar 5.2: penyederhanakan TG

Transition graph, disingkat TG, adalah kumpulan dari tiga hal:

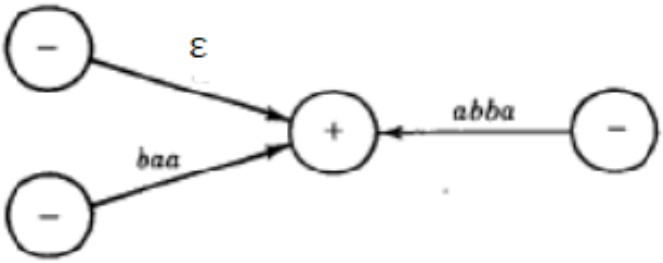
a. Himpunan state berhingga setidaknya satu di antaranya ditetapkan sebagai state awal (-) dan beberapa (mungkin tidak ada) yang ditetapkan sebagai state akhir (+)., 2. Alfabet Σ dari kemungkinan sebagai huruf input dan akan terbentuk menjadi string. 3. Himpunan transisi terbatas yang menunjukkan bagaimana berpindah dari satu state ke state lainnya membaca substring tertentu dari huruf input (bahkan mungkin string kosong λ)

   Perhatikan TG berikut:



Gambag 5.3: Mesin TG menerima abb Ɛa, abbƐaab

Contoh ; Setiap TG di mana beberapa state awal juga merupakan state akhir menerima λ, ini juga berlaku untuk FA. Ada beberapa TG lain yang menerima kata Ɛ, misalnya:



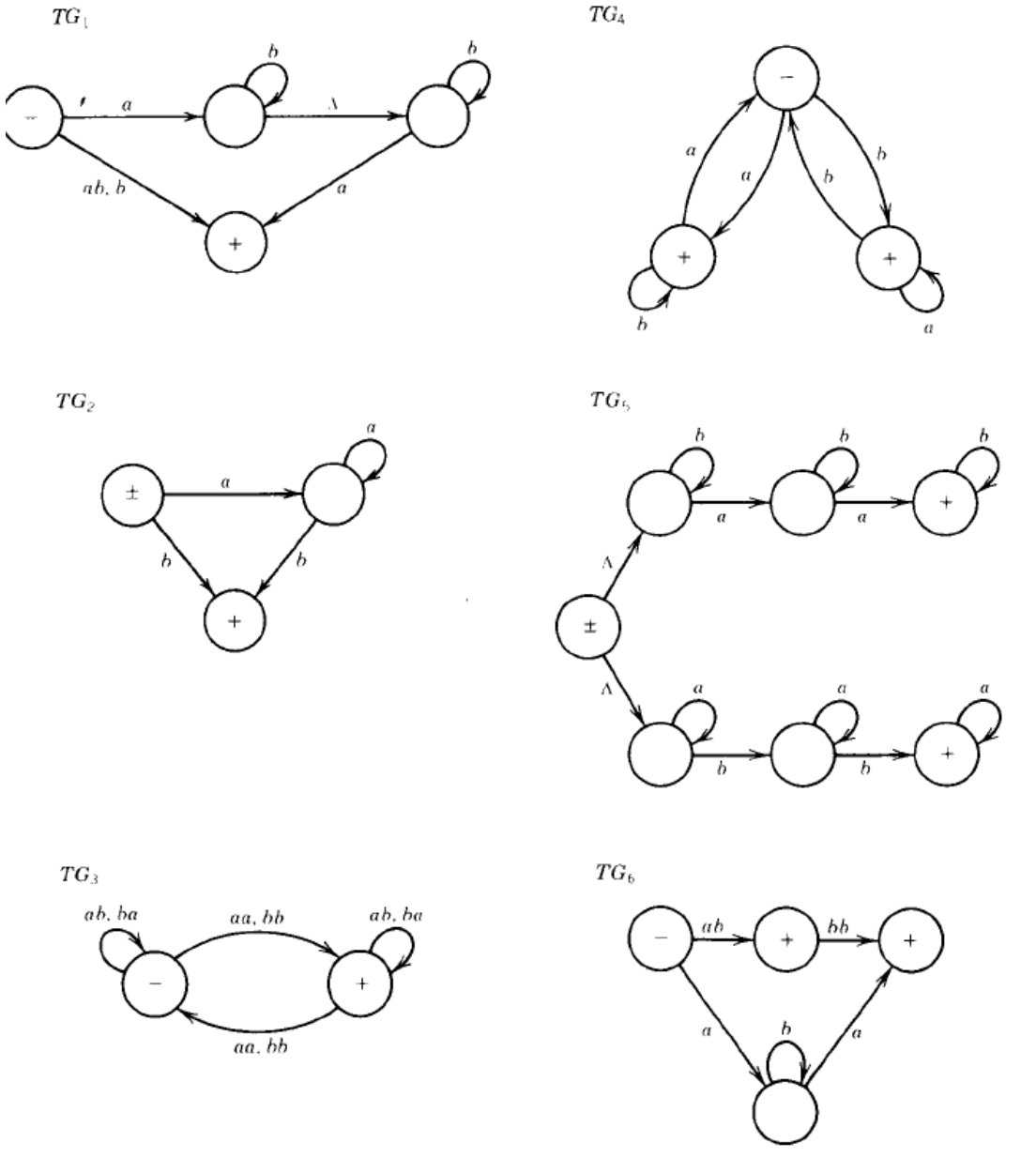
Gambar 5.4: mesin TG memiliki beberapa State awal

Mesin ini pada gambar 4.4. hanya menerima kata Ɛ , baa, dan abba.

Latihan-latihan soal

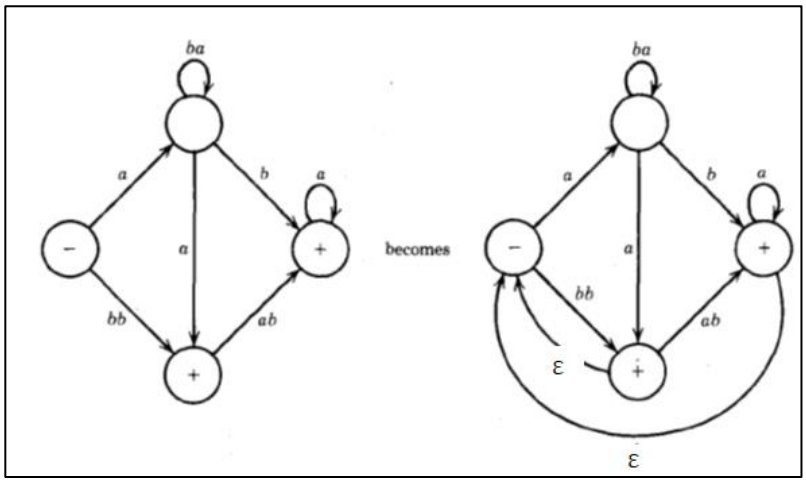
Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

a. Berikut enam mesin TG. Untuk masing-masing dari 10 kata berikutnya,tunjukkan yang mana dari mesin menerima kata yang diberikan.
   1) Ɛ
   2) aba,
   3) a
   4) abba,
   5) b
   6) bab
   7) aa
   8) baab
   9) ab
   10) abbb



Gambar 5.5: Enam Mesin TG

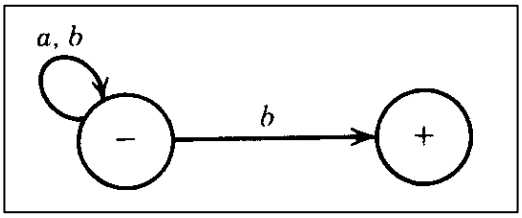
b. Ditunjukkan TG sebagai berikut, jelaskan dengan bahasa "L" yang diterima, yang awal mesin dan yang kedua setelah perubahan mesin setelah diberikan inputan null string ?



Gambar 5.6: Mesin TG

CONTOH

Perhatikan TG berikut ini:



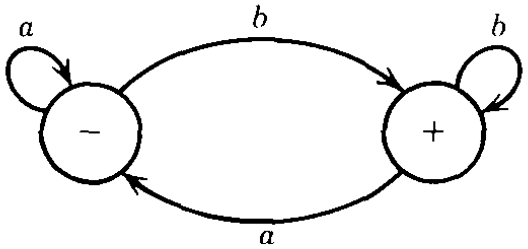
Gambar 5.7: TG

Kita dapat membaca semua huruf yang dimasukkan satu per satu dan tetap berada di sisi kiri state. Ketika kita membaca a b dalam state ada dua kemungkinan sisi yang kita dapat mengikuti. Jika huruf terakhir adalah a b, kita dapat menggunakannya untuk menuju ke state +. Itu pasti huruf terakhir, karena sekali dalam state sisi kanan jika kita mencoba untuk membaca kata lain kita akan crash.

Perhatikan bahwa juga mungkin untuk memulai dengan kata yang diakhiri dengan a b tetapi untuk mengikuti jalan yang gagal yang tidak mengarah pada state penerimaan. Kita bisa membuat kesalahan dengan mengikuti b non-loop terlalu cepat (bulan final pada input b) dalam hal ini kita gagal pada huruf berikutnya; atau kita mungkin membuat kesalahan dengan mengulang kembali ke - ketika kita membaca b terakhir, di mana kasus kami menolak tanpa crash. Tapi tetap saja, semua kata yang berakhiran b bisa jadi diterima, dan hanya itu yang diperlukan. Bahasa yang diterima oleh TG ini adalah semua kata yang berakhiran b. Satu reguler ekspresi untuk bahasa ini adalah

$$(a + b)^*b$$

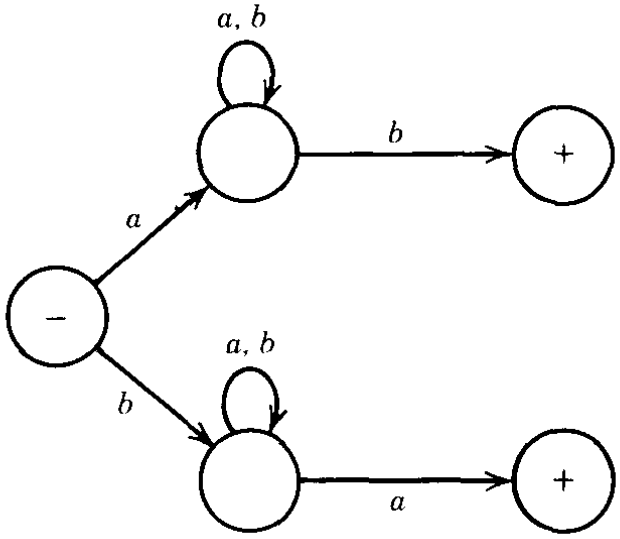
dan FA yang menerima hal yang sama bahasa adalah:



Contoh

TG berikut ini:

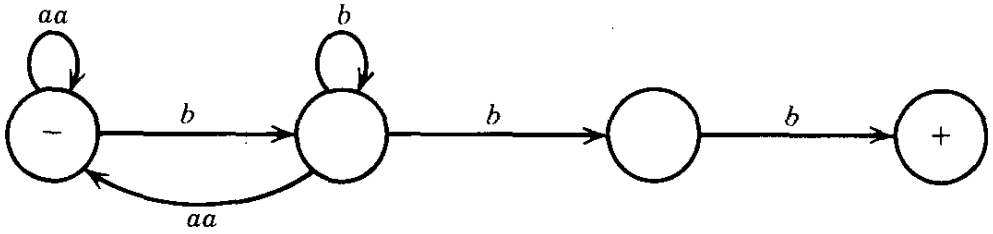
Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress



menerima bahasa semua kata yang dimulai dan diakhiri dengan huruf yang berbeda.

Contoh

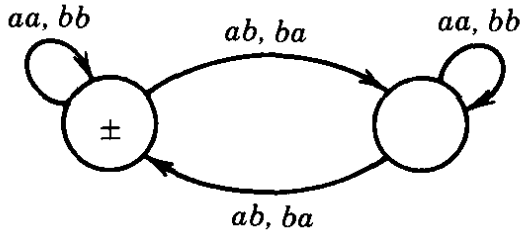
TG berikut ini:



menerima bahasa semua kata di mana a hanya muncul dalam rumpun genap dan yang diakhiri dengan tiga atau lebih b.

CONTOH

TG berikut ini:



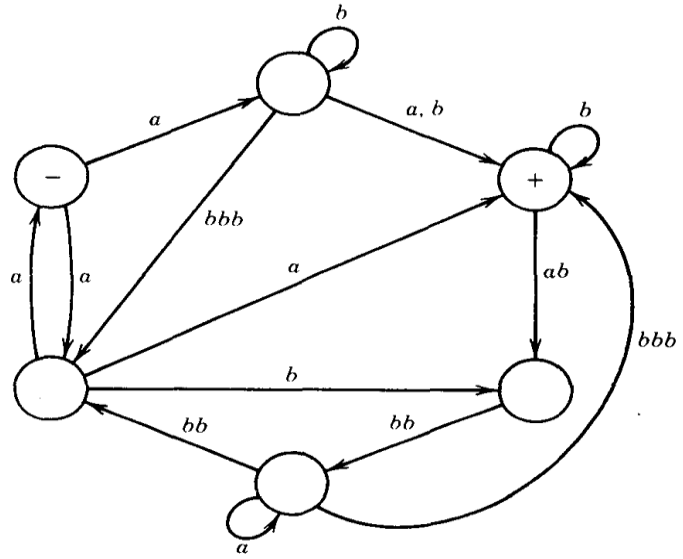
Dalam TG ini setiap tepi diberi label dengan sepasang huruf. Ini berarti bahwa untuk string yang akan diterima harus memiliki jumlah huruf genap yang dibaca dalam dan diproses dalam kelompok dua. Mari kita sebut state kiri seimbang state dan state kanan state tidak seimbang. Jika pasangan huruf pertama itu kita baca dari input string adalah double (aa atau bb), maka mesin tetap dalam state seimbang. Dalam keadaan seimbang mesin telah membaca genap jumlah a dan jumlah b genap. Namun, ketika sepasang tak tertandingi huruf dibaca (baik ab atau ba), mesin membalik ke state tidak seimbang yang menandakan bahwa ia telah membaca bilangan ganjil dari a dan bilangan ganjil dari b. Kami tidak kembali ke keadaan seimbang sampai "sesuai" lain yang tak tertandingi

pasangan dibaca (tidak harus pasangan yang tidak cocok, setiap tidak sama pasangan). Penemuan dua pasangan yang tidak sama membuat jumlah total a dan jumlah total b yang dibaca dari string input bahkan lagi. TG ini adalah contoh mesin yang menerima bahasa lama dan sangat familiar GENAP-GENAP semua kata dengan bilangan genap a dan bilangan genap dari b.

Dari ketiga contoh definisi atau deskripsi bahasa ini kami memiliki (ekspresi reguler, FA, dan TG); yang terakhir ini yang paling dimengerti. Ada masalah praktis dengan TG. Terkadang ada begitu banyak kemungkinan cara mengelompokkan huruf dari string input yang harus kita periksa banyak kemungkinan sebelum kita mengetahui apakah string yang diberikan diterima atau ditolak.

Contoh

Pertimbangkan TG ini:



Apakah kata abbbabbbabba diterima oleh mesin ini? (Ya, dalam dua cara.)

Ketika kita mengizinkan Ɛ -edges, kita mungkin memiliki jumlah cara pengelompokan yang tak terbatas huruf dari string input. Misalnya, string input ab mungkin difaktorkan sebagai:

(a) (b)

(a) (Ɛ) (b)

(a) (Ɛ) (Ɛ) (b)

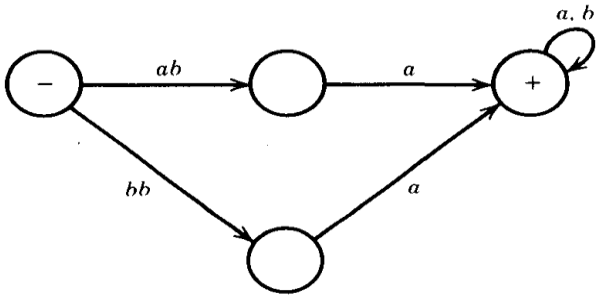
(a) (Ɛ) (Ɛ) (Ɛ) (b)

string tertentu diterima oleh TG tertentu, kami akan menunggu sampai Bab sebelumnya ketika tugas akan lebih mudah. Tentu saja ada algoritma yang sulit untuk melakukan tugas ini yang berada dalam kemampuan kita saat ini. Salah satunya adalah diuraikan dalam Soal 20 di bawah ini.

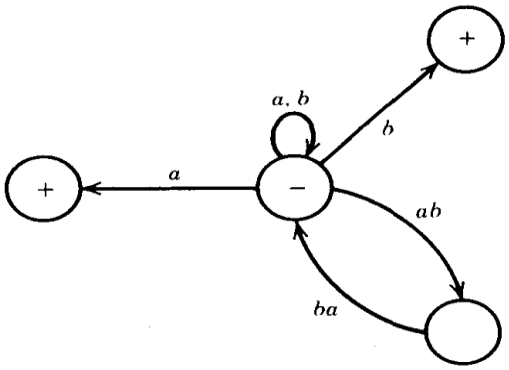
5.2 Latihan-Latihan Soal

Untuk TG transisi berikut, selesaikan denan algoritma ekspresi reguler yang setara, Kemudian sederhanakan ekspresi tersebut jika memungkinkan.

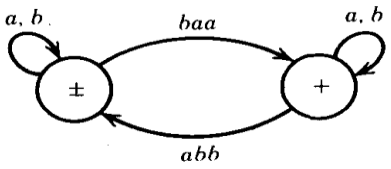
2.



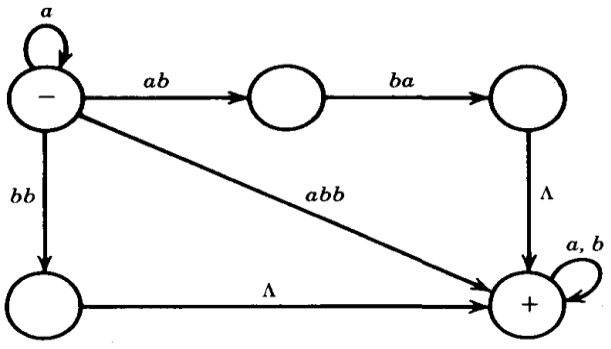
3.



4.



5.



KLEENE'S THEOREM

Dalam bab sebelumnya diperkenalkan tiga cara secara terpisah untuk mendefinisikan bahasa,ekspresi reguler,dan finite automata. Ingat bahwa bahasa yang didefinisikan oleh mesin adalah kumpulan dari semua kata-kata yang diterima oleh FA.) Dalam bab ini akan disajikan teorema yang dibuktikan oleh Kleene pada tahun 1956, yang mengatakan bahwa jika suatu bahasa dapat didefinisikan oleh salah satu dari ketiga cara ini, maka dapat juga didefinisikan dengan dua cara lainnya. Salah satu cara menyatakan ini untuk mengatakan bahwa ketiga metode mendefinisikan bahasa ini adalah setara.

Teorema

Bahasa yang dapat didefinisikan oleh

1) Ekspresi reguler, atau
2) Finite automata, atau
3) Transsition Graph,

Bahasa dapat didefinisikan oleh ketiga metode tersebut.

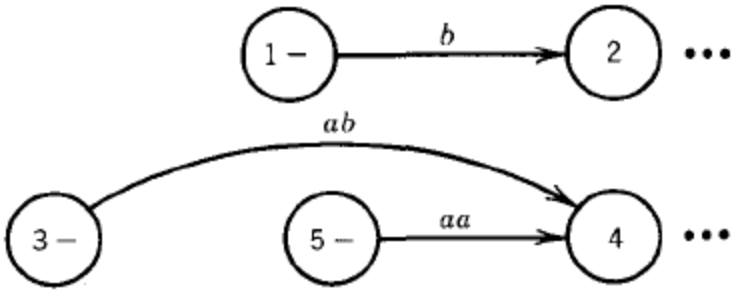
Bukti

1) Setiap bahasa yang dapat didefinisikan oleh FA juga dapat ditentukan oleh TG.
2) Setiap bahasa yang dapat didefinisikan oleh TG juga dapat didefinisikan oleh ekspresi reguler.
3) Setiap bahasa yang dapat didefinisikan dengan ekspresi reguler juga dapat didefinisikan oleh FA.

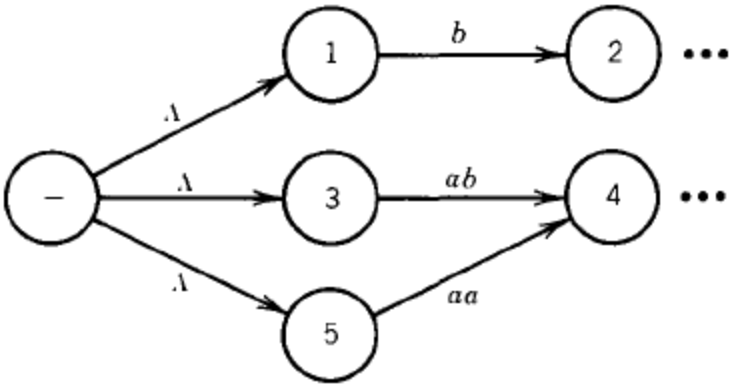
Setiap FA itu sendiri adalah TG, oleh karena itu, bahasa apa pun yang telah ditentukan oleh FA juga telah didefinisikan oleh TG. Demikian pula bukti yang lain, bukti dari bagian ini akan dengan algoritma konstruktif. Ini berarti bahwa kita menyajikan FA dan berakhir dengan ekspresi reguler yang mendefinisikan bahasa yang sama. Artinya sebaliknya dari FA kita uraikan dengan algoritma konstruktif menjadi ekspresi regular, dan kemudian juga tentu akan mendefinisikan bahasa yang sama.

Contoh 1.

Pada gambar 5.1. adalah diilustrasikan pada TG yang memiliki tiga state awal: 1, 3, dan 5



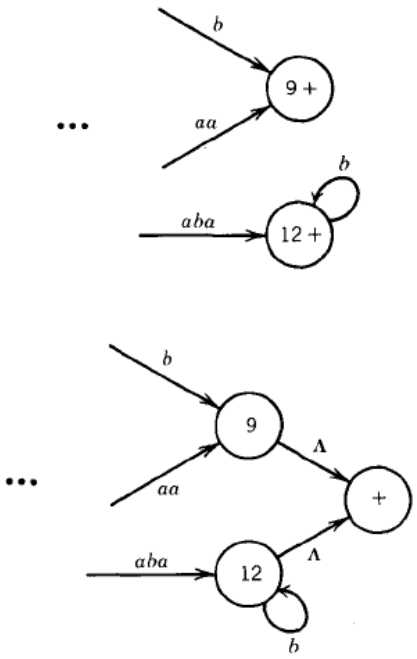
Menjadi



Gambar 5.8: Perubahan bentuk Mesin FA

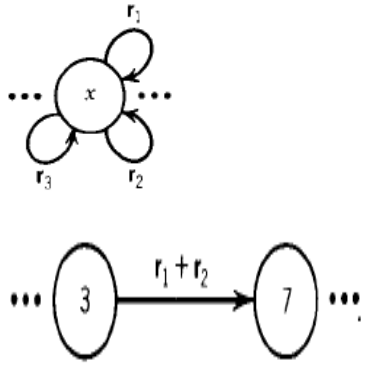
Contoh 2. Pada gambar

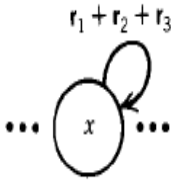
Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

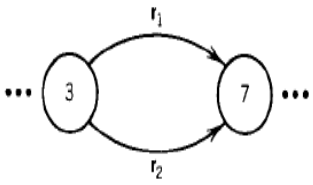


Gambar 5.9: Perubahan tempat state akhir

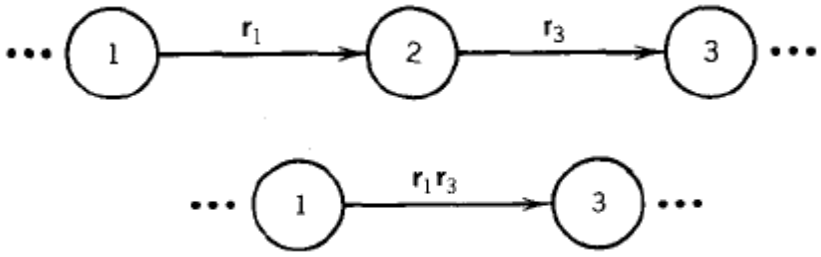
Gambar ekpresi regular

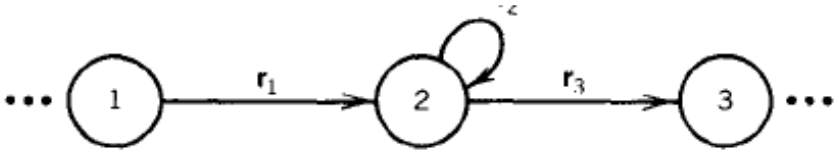




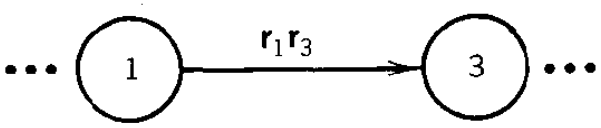


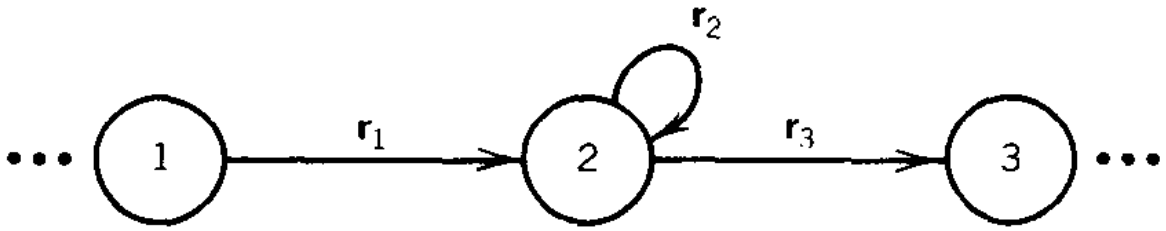
(a) r1r2r3 (b) r1+r2+r3 (c). R1+r2 (d). r1+r2



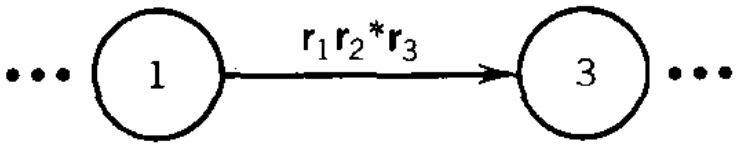


Menjadi





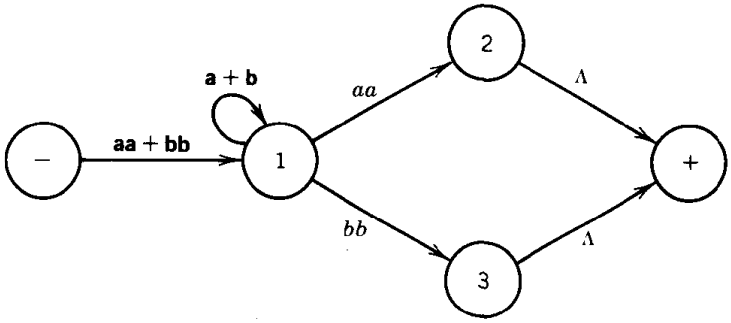
Menjadi

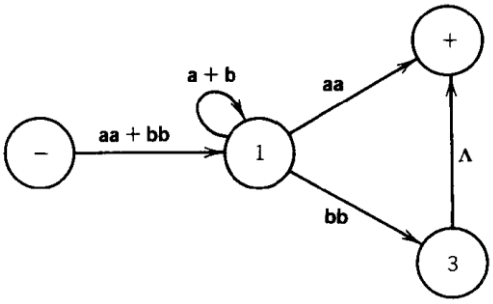


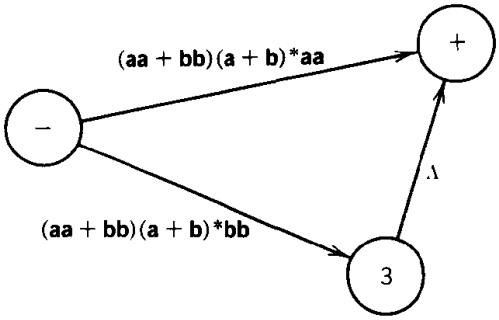
Kita membuktikan pada bagian 2 Setiap bahasa yang dapat didefinisikan oleh TG juga dapat didefinisikan oleh ekspresi reguler.

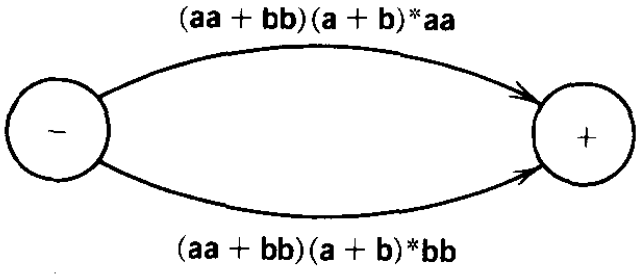
Contoh 5.1.

Ditunjukan Transition graph sebagai berikut









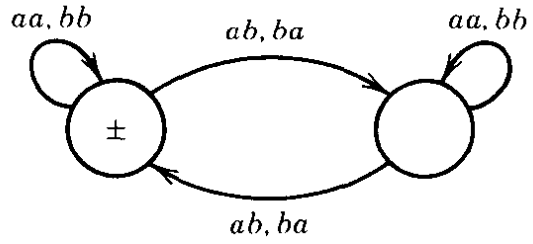
Dengan demikian maka bagian ke dua terbukti bahwa mesin ini mendefinisikan bahasa yang sama dengan ekspresi reguler

(aa + bb)(a + b)*(aa) + (aa + bb)(a + b)*(bb)

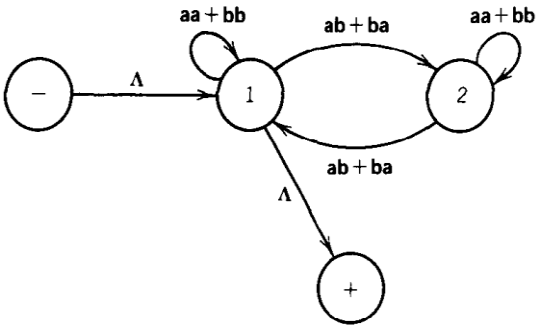
Contoh 5.2.

Ditunjukkan bagaimana proses didefinisikan TG dengan regular ekspresi

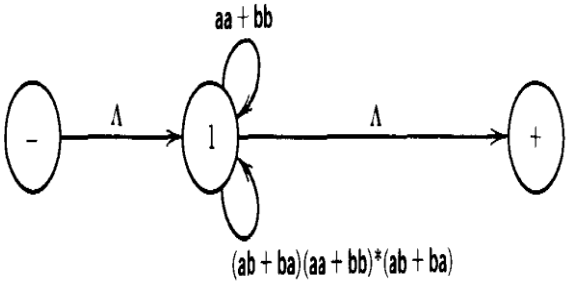
[(aa + bb) + (ab + ba) (aa + bb)*(ab + ba)]* sebagaimana proses pada gambar a,b,c,d dan menghasilkan gambar e



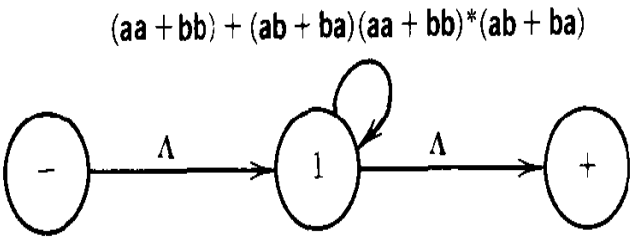
(a)



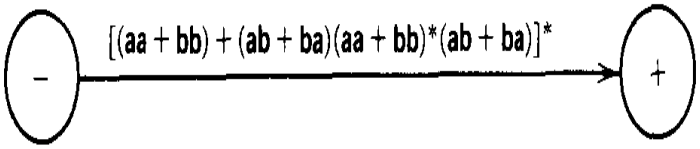
(b)



(.c)

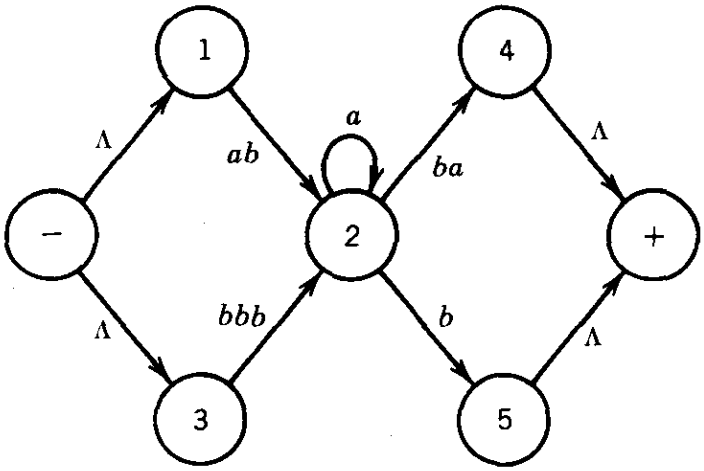


(d)



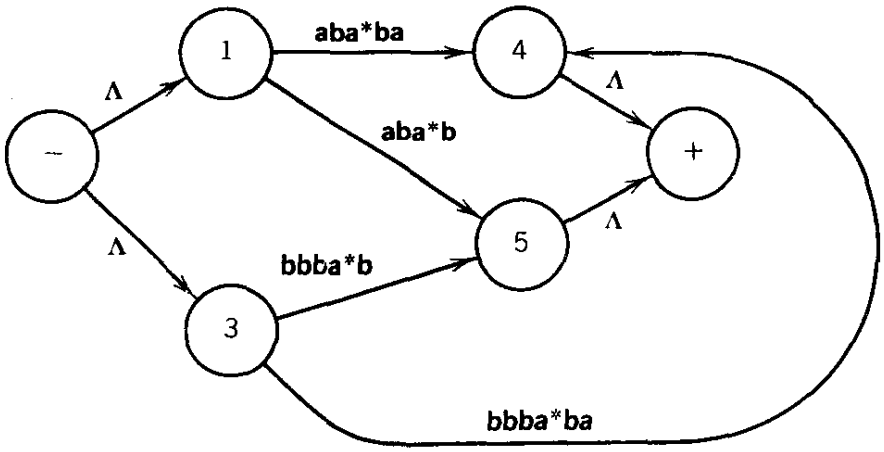
(e)

Misalnya, jika kita memiliki:

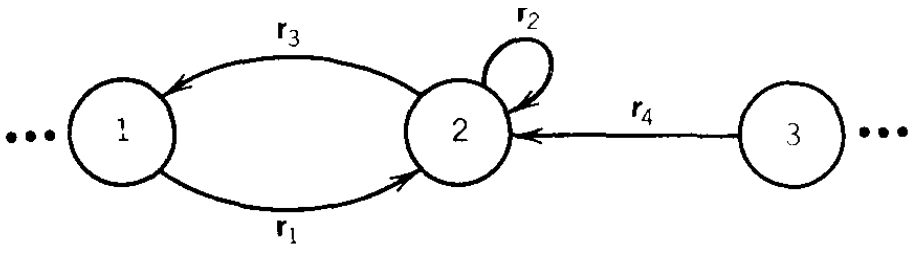


kita dapat melewati state 2 dengan memasukkan jalur dari state 1 ke state 4 berlabel aba*ba, jalur dari state 1 ke state berlabel aba*b, jalur dari state 3 untuk menyatakan 4 berlabel bbba*ba, dan jalur dari state ke state 5 berlabel bbba*b.Kami kemudian dapat menghapus tepi dari state 1 ke state 2 dan dari state 3 ke state 2. Tanpa edge ini, state 2 menjadi unreachable. Tepi dari state 2 ke state 4 dan 5 kemudian tidak berguna karena mereka tidak dapat menjadi bagian dari jalur dari - ke +. Pada state ini dan sisi ini tidak akan memengaruhi apakah kata apapun yang diterima oleh TG ini.

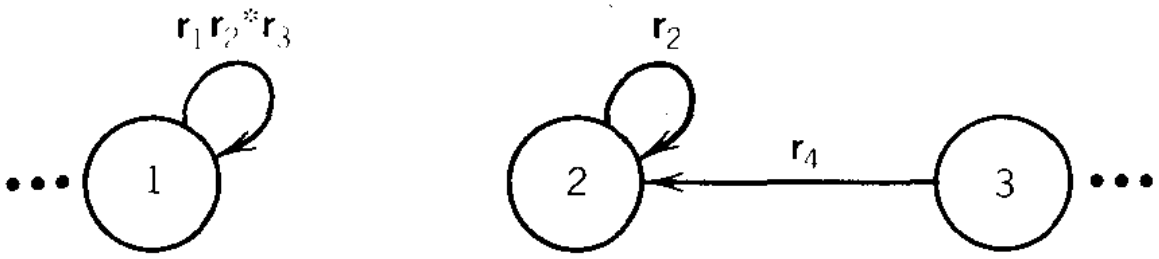
Mesin yang dihasilkan dari operasi ini adalah:



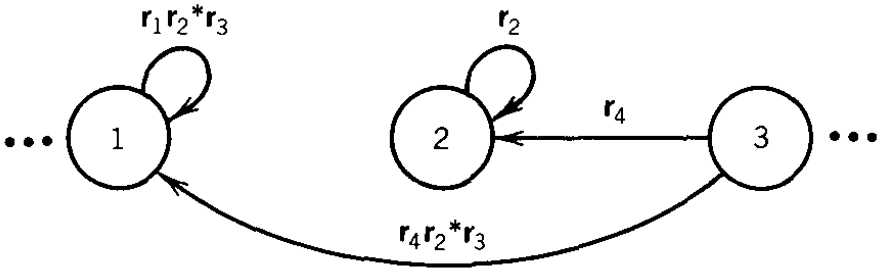
Sebelum selesai menjelaskan algoritma ini, ada beberapa: kasus khusus yang harus kita kaji lebih teliti. Dalam gambar dibawah ini



menjadi



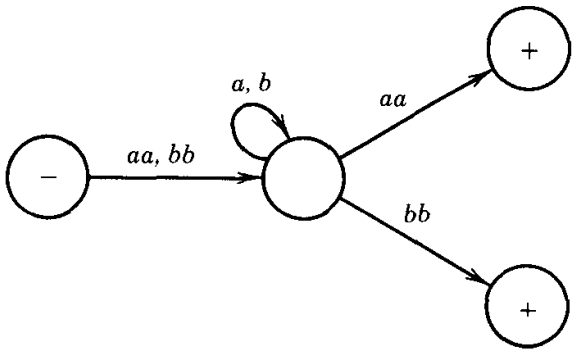
Menjadi



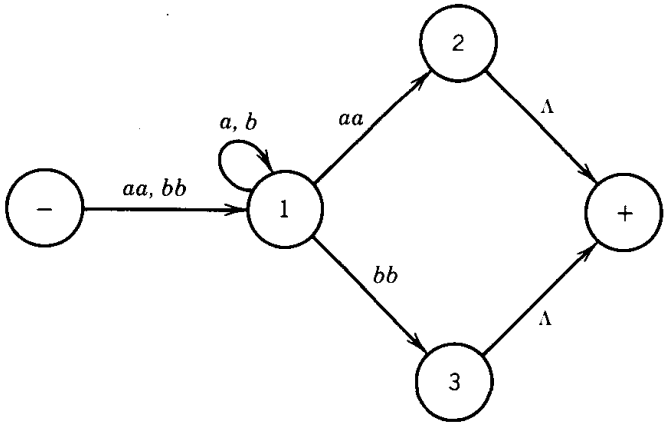
Setiap kali kita menghilangkan edge atau state, kita harus yakin bahwa kita memilikinya tidak menghilangkannya jalur apa pun melalui T yang mungkin telah ada sebelumnya. Mehilangkan atau menghapus bisa mengubah bahasa kata-kata yang diterima, yang tidak kita inginkan melakukan. Karena hanya ada banyak jalur di seluruh TG (tidak termasuk jalur perulangan dan pengulangan), kita dapat memeriksa kemungkinan ini dalam jumlah yang terbatas. Contoh ini merupakan gejala dari satu-satunya masalah yang muncul dengan algoritma ini, jadi sekarang memiliki metode yang dijelaskan dengan baik untuk menghasilkan ekspresi reguler

Yang ekuivalen dengan gambar transisi yang diberikan. Semua kata yang diterima oleh T adalah melalui gambar T. Jika kita mengubah gambar tetapi mempertahankan semua jalur dan label mereka, kita harus menjaga bahasa tidak berubah. Algoritma ini berakhir dalam jumlah langkah yang terbatas, karena T hanya memiliki banyak state untuk memulai, dan satu state dihilangkan dengan setiap iterasi. Pengamatan penting lainnya adalah bahwa metode ini bekerja pada semua gambar transisi Oleh karena itu, algoritma ini memberikan bukti yang memuaskan bahwa ada ekspresi reguler untuk setiap grafik transisi. Sebelum melanjutkan ke pembuktian Bagian 3, mari kita ilustrasikan algoritmanya di atas pada contoh tertentu.

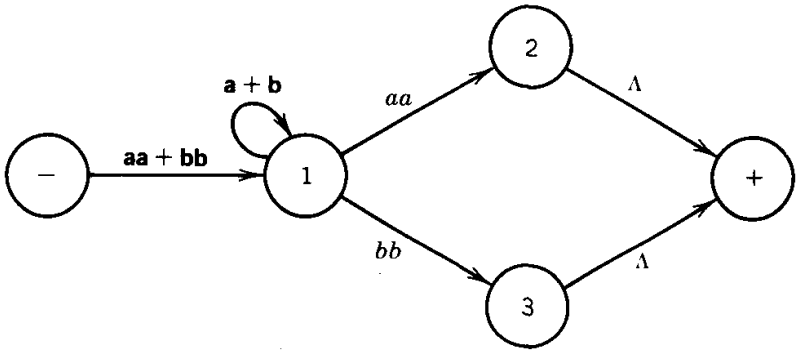
TG yang akan kami pertimbangkan adalah yang di bawah ini, yang menerima semua kata yang dimulai dan diakhiri dengan huruf ganda (memiliki setidaknya empat huruf berbeda). Ini bukanlah satu-satunya TG yang menerima bahasa ini.



mesin ini hanya memiliki satu status awal, tetapi memiliki dua status akhir, jadi kita harus memperkenalkan state akhir unik baru mengikuti metode yang ditentukan dengan algoritma

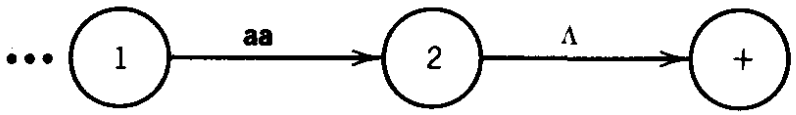


Modifikasi selanjutnya yang kita lakukan adalah dengan memperhatikan edge dari awal state ke state 1 adalah sisi ganda kita dapat melewati aa atau bb. Kami menggantinya dengan ekspresi reguler aa + bb. Kami juga mencatat bahwa ada loop ganda pada state 1. Kita dapat mengulang kembali ke state 1 pada satu a atau pada satu b. Algoritme mengatakan kita seharusnya mengganti loop ganda ini oleh satu loop berlabel dengan ekspresi reguler a + b. gambar dari mesin sekarang menjadi:

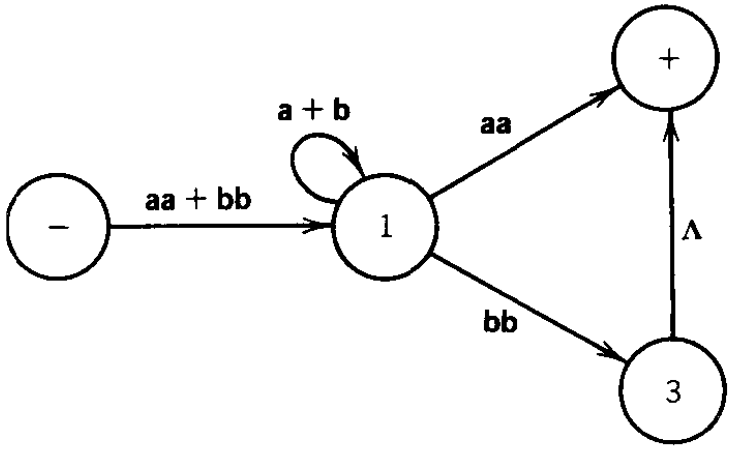


Mari kita pilih untuk modifikasi selanjutnya jalur dari state 1 ke state 2 menyatakan +. Algoritme tidak benar-benar memberi tahu kami bagian mana dari TG kita harus menyerang selanjutnya. Urutan diserahkan kepada kebijaksanaan kita sendiri. Algoritme memberitahu kita bahwa itu benar-benar tidak masalah. Selama kita terus menghilangkan sisi dan state, kami akan menyederhanakan mesin menjadi satu reguler representasi dari ekspresi.

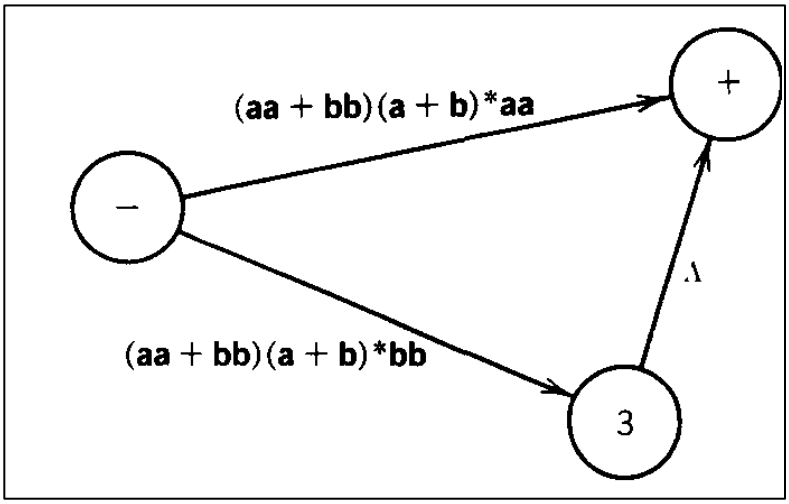
Perhatikan dan pertimbangkan graph ini,



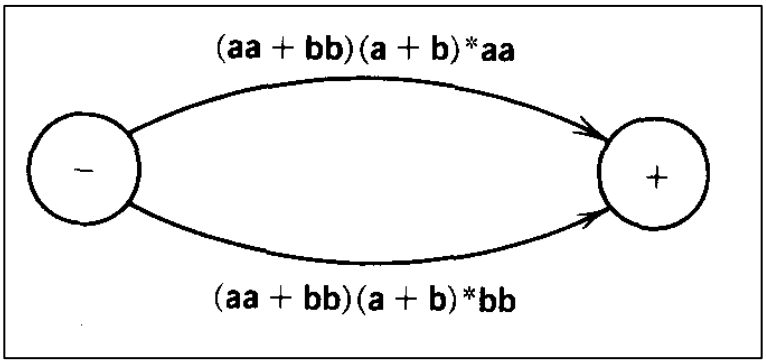
Dari gambar transisi diatas kita dapat buat gambar transisi sebagaimana berikut



Mari kita coba untuk melewati State 1. Hanya satu sisi yang masuk ke State keadaan 1 dan itu adalah dari State. Ada loop pada state 1 dengan label (a + b). State 1 memiliki sisi yang keluar darinya yang mengarah ke State 3 dan State +. Algoritma menjelaskan bahwa kita dapat menghilangkan State 1 dan mengganti sisi ini dengan sisi dari State - ke State 3 berlabel (aa + bb)(a + b)*(bb) dan tepi dari State - ke State + berlabel (aa + bb)(a + b)*(aa). Yang berbintang ekspresi di tengah mewakili fakta bahwa saat kita berada di State 1 kita dapat berputar-putar selama yang kita inginkan, dua kali atau bahkan tidak ada samasekali. Bukan kebetulan bahwa definisi operator penutupan * persis sesuai dengan situasi perulangan, karena Kleene menciptakannya untuk tujuan ini. Setelah menghilangkan State 1, mesin terlihat seperti ini:



Jelas bahwa kita sekarang harus menghilangkan state 3, karena itu adalah satu-satunya state yang dapat dilewati yang tersisa. Saat kita menggabungkan ekspresi reguler dari state - untuk menyatakan 3 dengan ekspresi reguler dari state 3 ke state +, mesin:



Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

Sekarang dengan aturan terakhir dari algoritma, mesin ini mendefinisikan bahasa yang sama sebagai ekspresi reguler,

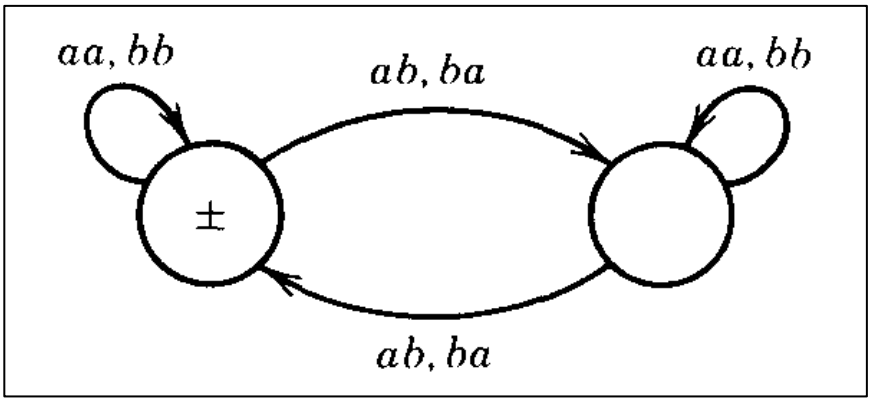
$$(aa + bb)(a + b)^*(aa) + (aa + bb)(a + \mathbf{b})^*\mathbf{(bb)}$$

Jika kita harus membuat ekspresi reguler untuk bahasa semua string yang dimulai dan diakhiri dengan huruf ganda, kita mungkin akan menulis:

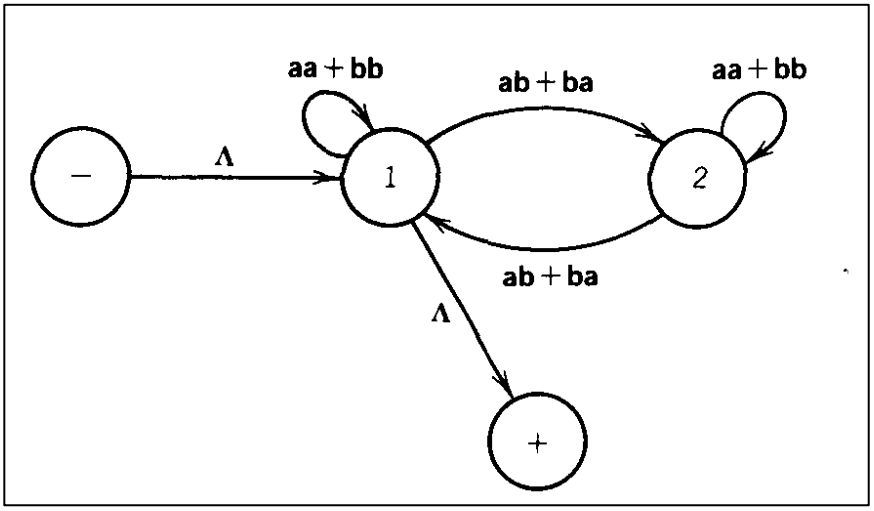
$$(aa + bb)(a + b)^*(aa + \mathbf{bb})$$

yang setara dengan ekspresi reguler yang dihasilkan algoritma karena hukum distributif aljabar berlaku untuk ekspresi reguler. Tanpa melalui deskripsi panjang lebar, mari kita lihat algoritmanya bekerja pada satu contoh lagi.

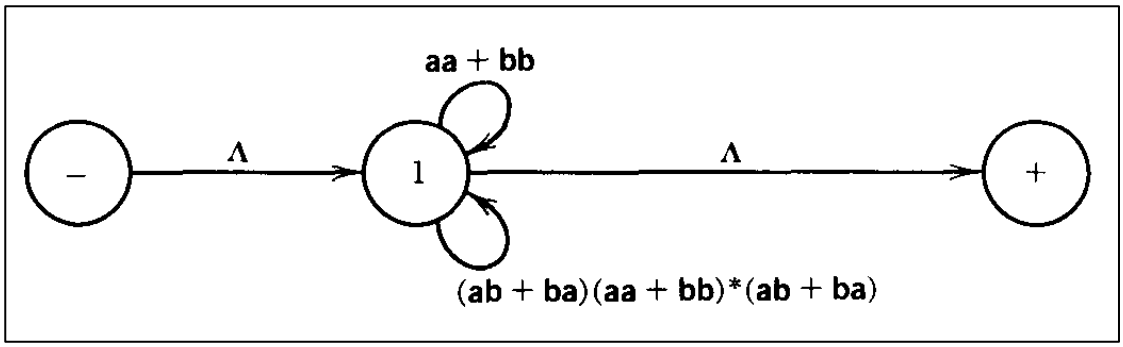
Mari kita mulai dengan TG yang menerima string dengan bilangan genap a dan bilangan genap b, bahasanya BAHKAN-GENAP.



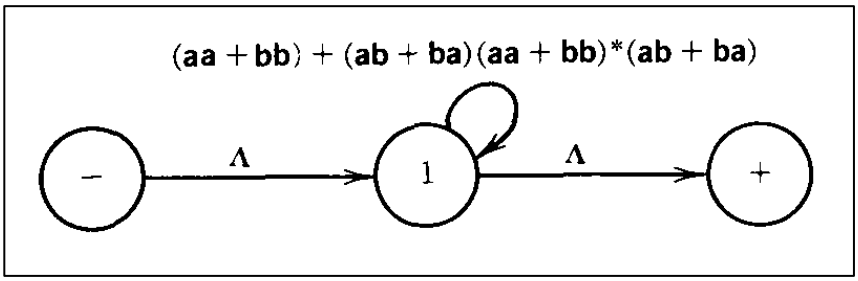
menjadi yang pertama



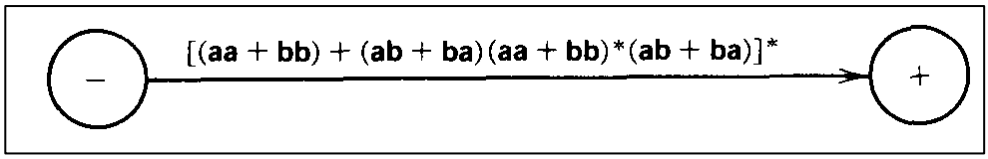
Ketika kita menghilangkan state 2, jalur dari 1 ke 2 ke I menjadi loop di state 1



yang menjadi:



yang menjadi:



yang direduksi menjadi ekspresi reguler:

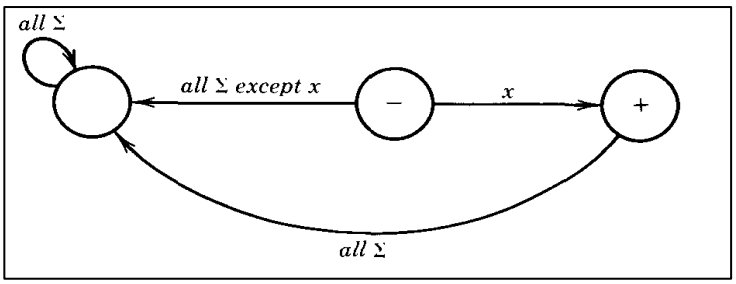
$$(aa + bb) + (ab + ba)(aa + bb)^*(ab + ba)]^*$$

Kami menyajikan algoritma kami secara rekursif.

Aturan 1

Ada FA yang menerima huruf alfabet tertentu. Di sana FA yang hanya menerima kata Ɛ.

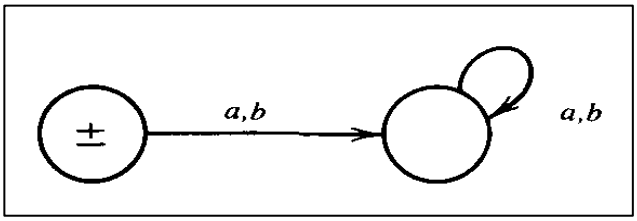
Contoh: Jika x dalam ∑, maka FA



hanya menerima kata x.

Satu FA yang hanya menerima Ɛ adalah

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress



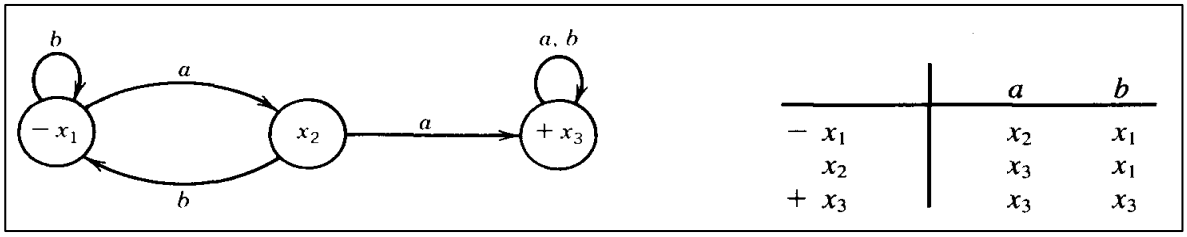
Aturan 2

Jika ada FA yang disebut FA1 yang menerima bahasa yang ditentukan oleh ekspresi reguler r, dan ada FA yang disebut FA2 yang menerima bahasa didefinisikan oleh ekspresi reguler r 2, maka ada FA yang disebut FA3 yang menerima bahasa yang ditentukan oleh ekspresi reguler (rl + r 2).

Kami akan membuktikan ini dengan menunjukkan cara membuat mesin baru dengan cara yang paling masuk akal. Sebelum kita menyatakan prinsip-prinsip umum, mari kita tunjukkan secara spesifik contoh. Misalkan kita memiliki mesin FA1, yang menerima bahasa semua kata di atas alfabet

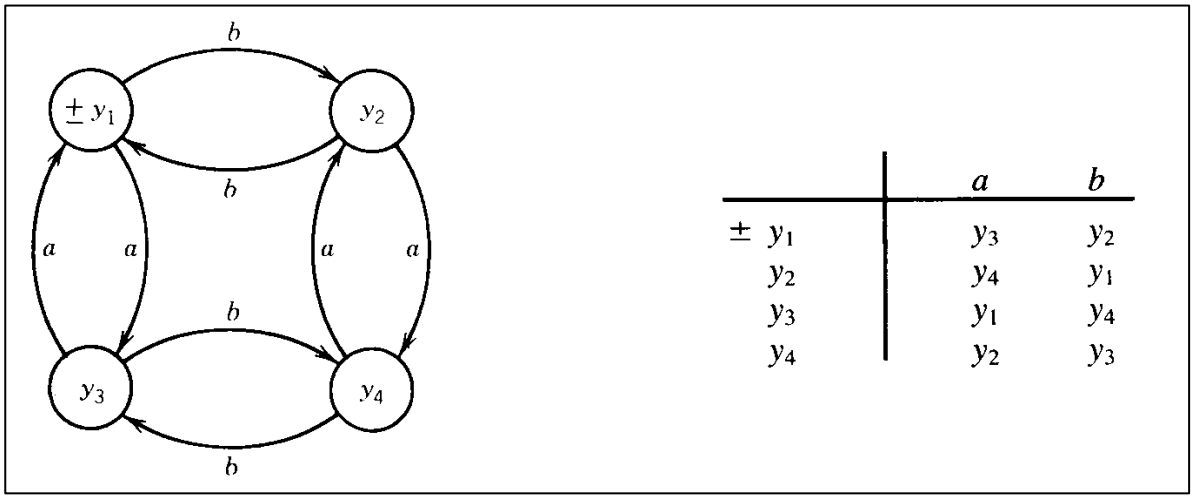
$$\sum = \{a, b\}$$

yang memiliki double a di suatu tempat di dalamnya,



	a	b
− x_1	x_2	x_1
x_2	x_3	x_1
+ x_3	x_3	x_3

dan mesin FA2 yang sudah dikenal, yang menerima semua kata yang memiliki keduanya genap jumlah total a dan bilangan genap dari total b



	a	b
± y_1	y_3	y_2
y_2	y_4	y_1
y_3	y_1	y_4
y_4	y_2	y_3

Kami akan menunjukkan bagaimana merancang mesin yang menerima kedua set. Artinya, kita akan membangun mesin yang menerima semua kata yang memiliki aa atau BAHKAN menolak semua string yang tidak memiliki karakteristik.

Bahasa yang diterima mesin baru akan menjadi gabungan dari keduanya bahasa. Kami akan memanggil State dalam mesin baru ini zi, z2 , z 3 , dan seterusnya, untuk sebanyak yang kita butuhkan. Kami akan mendefinisikan mesin ini dengan tabel transisinya. Prinsip panduan kami adalah ini: Mesin baru akan secara bersamaan menjaga melacak di mana inputnya jika dijalankan di FA1 dan di mana jika masukan akan berjalan di FA2.

Pertama-tama, kita membutuhkan jika awal. state ini harus menggabungkan x1 ,awal state untuk FAI, dan yl, state awal untuk FA2 . Kami menyebutnya z 1. Jika string adalah berjalan di FAI, itu akan dimulai di x1, di FA2 di y1.

State baru yang terjadi jika huruf input a dibaca? Jika string sedang dijalankan pada mesin pertama, itu akan menempatkan mesin ke state x2. Jika string berjalan pada mesin kedua, itu akan menempatkan mesin menjadi state Y3. Oleh karena itu, pada mesin baru kami a menempatkan ke dalam state 2 , yang berarti baik x2 atau Y3, dengan cara yang sama bahwa z, berarti baik x, atau yl. Sejak y, adalah state akhir, untuk FA2, Zl juga merupakan state akhir dalam arti bahwa kata apa pun yang jalurnya berakhir di sana pada mesin-z akan diterima oleh FA2.

$$z_1 = x_1 \text{ or } y_1$$

$$z_2 = x_2 \text{ or } y_3$$

Pada mesin FA3 kami mengikuti kedua jalur yang akan dibuat input di FA1 dan jalur di FA2 secara bersamaan. Dengan mengikuti kedua jalur tersebut, kita tahu kapan string input berakhir apakah sudah mencapai final atau belum state di kedua mesin.

Jika kita berada dalam state z, dan kita membaca huruf b, kita kemudian pergi ke state z3, yang mewakili baik x, atau Y2. X, berasal dari berada di x, di FA, dan membaca a b, sedangkan Y2 berasal dari berada di yj pada FA2 dan membaca a b.

Z3 = x, or Y2

Awal dari tabel transisi untuk FA3 adalah:

	a	*b*
± z_1	z_2	z_3

Misalkan entah bagaimana kita telah masuk ke state z2 dan kemudian kita membaca a.Jika kita berada di FAI, kita sekarang akan pergi ke state x3, yang merupakan state akhir. Jika kami berada di FA2, sekarang kami akan kembali ke yi, yang juga merupakan state terakhir. Kami akan menyebut kondisi ini z4 , yang berarti x3 atau yl. Karena string ini bisa sekarang diterima di salah satu dari dua mesin ini, z4 adalah state akhir untuk FA3. Ternyata, dalam contoh ini kata tersebut diterima oleh kedua mesin di sekali, tapi ini tidak perlu. Penerimaan oleh mesin FA1 atau FA2 adalah cukup untuk diterima oleh FA3. Jika kita berada dalam state z2 dan kita membaca a b, maka dalam FA1 kita adalah kembali ke x, sedangkan di FA2 kita di Y4. Sebut kondisi baru ini z5 = menyatakan x, atau Y4.

$$+ \; z_4 = x_3 \text{ or } y_1$$

$$z_5 = x_1 \text{ or } y_4$$

Pada titik ini tabel transisi kita terlihat seperti ini:

	a	*b*
± z_1	z_2	z_3
z_2	z_4	z_5

Apa yang terjadi jika kita mulai dari state z3 dan membaca a? Jika kita berada di FAI, kita sekarang di x2, jika di FA2, kita sekarang di Y4. Ini adalah state baru; sebut saja state z6.

Z6 = X2 or y4

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

Bagaimana jika kita berada di z3 dan kita membaca a b? Di FAI kami tinggal di x1 , sedangkan di FA2 kita kembali ke yi. Ini berarti bahwa jika kita berada di z3 dan kita membaca a b kemudian kembali ke keadaan z1.

Tabel transisinya sekarang terlihat seperti ini:

		a	b
+	z_1	z_2	z_3
	z_2	z_4	z_5
	z_3	z_6	z_1

Bagaimana jika kita berada di z4 dan kita membaca a? FA1 tetap di x3, sedangkan FA2 pergi ke Y3. Ini adalah negara bagian baru; sebut saja z7. Jika kita berada di z4 dan kita membaca a b, bagian FA1 tetap di x3 sedangkan bagian FA2 pergi ke Y2. Ini state baru ; yaitu state z8.

$$+\ z_7 = x_3 \text{ or } y_3$$

$$+\ z_8 = x_3 \text{ or } y_2$$

Keduanya adalah state akhir karena string berakhir di sini pada mesin-z akan diterima oleh FAI, karena x3 adalah status final untuk FA1 . Jika kita berada di z5 dan kita membaca a, kita pergi ke x2 atau Y2, yang akan kita sebut Z9 . Jika kita berada di z5 dan kita membaca a b, kita pergi ke x, atau Y3, yang akan kita sebut z10 .

Z9 = x2 or Y2

zlo= x, or Y3

Jika kita berada di Z6 dan kita membaca a, kita pergi ke x3 atau Y2, yang merupakan Z8 lama

Jika kita berada di z6 dan kita membaca a b, kita pergi ke x, atau Y3, yaitu z10 lagi.

Jika kita berada di z7 dan kita membaca a kita pergi ke x3 atau yi, yaitu z4 lagi.

Jika kita berada di z7 dan kita membaca a b, kita pergi ke x3 atau Y4, yang merupakan state baru,Z11.

+ Z11 = X 3 or Y4

Jika kita berada di z8 dan kita membaca a, kita pergi ke x3 atau Y4 = z1 . Jika di z8 kita membaca a b, kita pergi ke x3 atau y1 = Z4

Jika kita berada di z9 dan kita membaca a, kita pergi ke x3 atau Y4 = Z11. Jika di z9 kita membaca a b, kita pergi ke x, atau y, = z1.

Jika kita berada di z10 dan kita membaca a, kita pergi ke x2 atau yl, state yang baru terakhir yaitu z 12.

+ Z12 = x2 or y1

Jika kita berada di z10 dan kita membaca a b, kita pergi ke (xi atau y4) z5 .

Jika kita berada di z11 dan kita membaca a, kita pergi ke (x3 atau y2) z 8.

Jika kita berada di zil dan kita membaca a b, kita pergi ke (x3 atau y3) Z7.

Jika kita berada di z12 dan kita membaca a, kita pergi ke (x3 atau Y3) = Z7.

Jika kita berada di Z12 dan kita membaca a b, kita pergi ke (x1 atau y2) z3.

Mesin yang sekarang dan Tabel transisi lengkapnya adalah:

		a	b
±	z_1	z_2	z_3
	z_2	z_4	z_5
	z_3	z_6	z_1
+	z_4	z_7	z_8
	z_5	z_9	z_{10}
	z_6	z_8	z_{10}
+	z_7	z_4	z_{11}
+	z_8	z_{11}	z_4
	z_9	z_{11}	z_1
	z_{10}	z_{12}	z_5
+	z_{11}	z_8	z_7
+	z_{12}	z_7	z_3

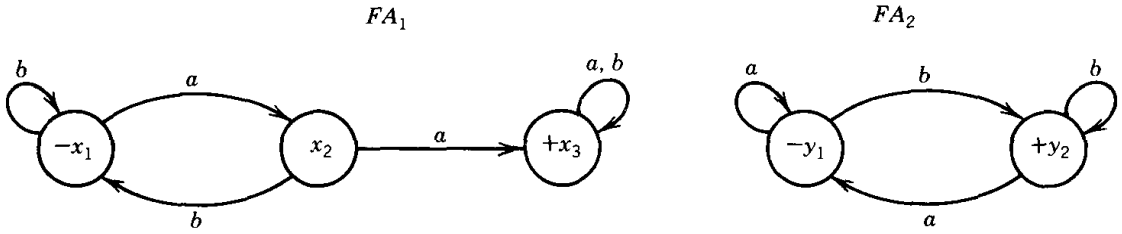
Jika sebuah string menelusuri mesin ini dan berakhir pada keadaan akhir, itu berarti bahwa itu juga akan berakhir pada keadaan akhir baik pada mesin FA1 atau pada mesin FA2. Juga, string apa pun yang diterima oleh FA1 atau FA2 akan diterima oleh ini FA3.

Gambaran umum dari algoritma yang kami gunakan di atas adalah sebagai berikut. Dimulai dengan dua mesin, FAI dengan status xi, x2, x3, . . . dan FA2 dengan negara bagian Y1, Y2, y3 . . ., membangun mesin baru FA3 dengan status zj, z2, z3. . .. di mana setiap z berbentuk "Xsomething atau Ysomething". Jika salah satu bagian x atau bagian y adalah keadaan akhir, maka z yang sesuai adalah keadaan akhir. Untuk pergi dari satu z ke yang lain dengan membaca huruf dari string input, kita melihat apa yang terjadi pada bagian x dan ke bagian y dan pergi ke z baru yang sesuai. Kita bisa menulis ini sebagai rumus:

Znew setelah huruf p = [Xnew setelah huruf p] atau [Ynew setelah huruf p]

Karena hanya ada banyak x dan y berhingga, maka hanya ada banyak banyak kemungkinan z. Tidak semuanya harus digunakan di FA3. Di dalam cara, kita dapat membangun mesin yang dapat menerima jumlah dari dua ekspresi reguler jika sudah ada mesin yang menerima setiap komponen secara teratur ekspresi secara terpisah.

Mari kita membahas ini dengan sangat cepat sekali lagi pada dua mesin:



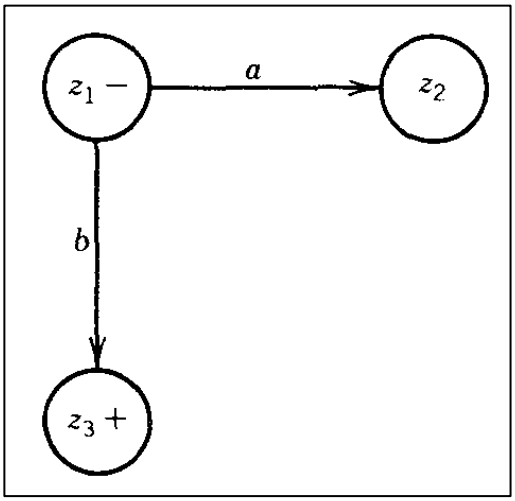
FAl menerima semua kata dengan double a, dan FA2 menerima semua kata berakhir di b. Mesin yang menerima penggabungan dua bahasa untuk keduanya mesin dimulai:

- z2 = x, atau Yj

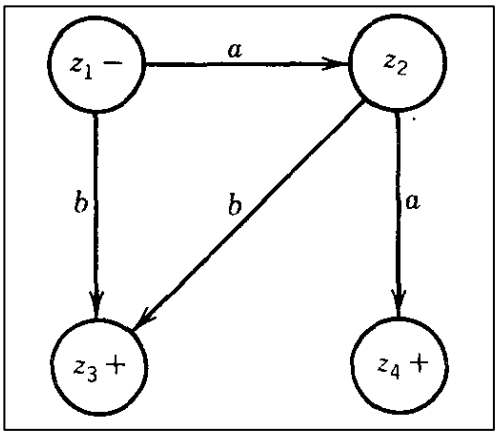
Di z, jika kita membaca a, kita pergi ke (X2 atau Yi) z2.

Dalam zl, jika kita membaca a b, kita pergi ke (x1 atau Y2) = z3, yang merupakan keadaan akhir karena Y2 adalah.

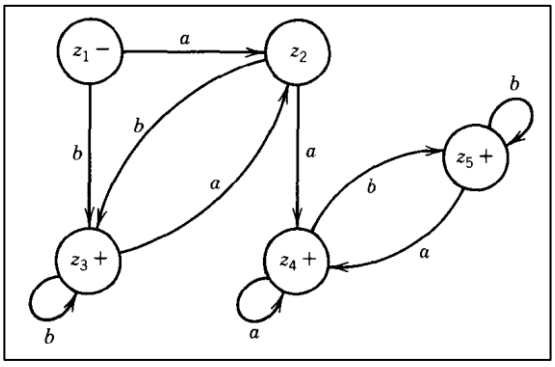
Gambaran sebagian dari mesin ini sekarang:



Dalam z2 jika kita membaca a, kita pergi ke (x3 atau yi) = z4 , yang merupakan state akhir karena x3 adalah.
Dalam z2 jika kita membaca a b, kita pergi ke (xi atau Y2) = z 3.



Dalam z3 jika kita membaca a, kita pergi ke (x2 atau yi) = z2.
Dalam z3 jika kita membaca a b, kita pergi ke (xi atau Y2) = z 3.
Di z4 jika kita membaca a, kita pergi ke (x3 atau yi) = z4.
Dalam z4 jika kita membaca a b, kita pergi ke (x3 atau Y2) z5 , yang merupakan keadaan akhir.
Di z5 jika kita membaca a, kita pergi ke (x3 atau yi) = z4.
Dalam z5 jika kita membaca a b, kita menuju ke x3 atau Y2 = z5.
Seluruh mesin terlihat seperti ini:



Mesin ini menerima semua kata yang memiliki double a atau yang berakhiran b.

Kemungkinan yang tampaknya logis

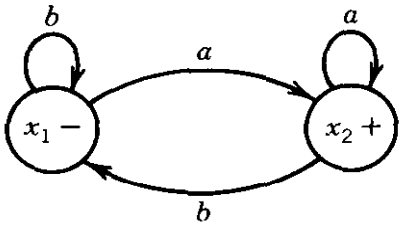
Z6= (x2 atau Y2)

tidak muncul. Ini karena berada di x2 di FA1 berarti huruf terakhir dibaca. Tapi berada di Y2 pada FA2 berarti huruf terakhir yang dibaca adalah a b. Ini tidak bisa keduanya benar pada saat yang sama, jadi tidak ada string input yang memiliki kemungkinan berada di state z6.

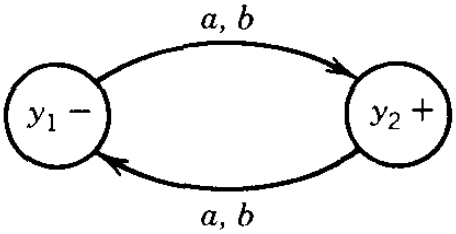
Algoritma ini menetapkan keberadaan mesin FA3 yang menerima penyatuan bahasa untuk FA1 dan FA2.

CONTOH (Dalam pembuktian Teorema)

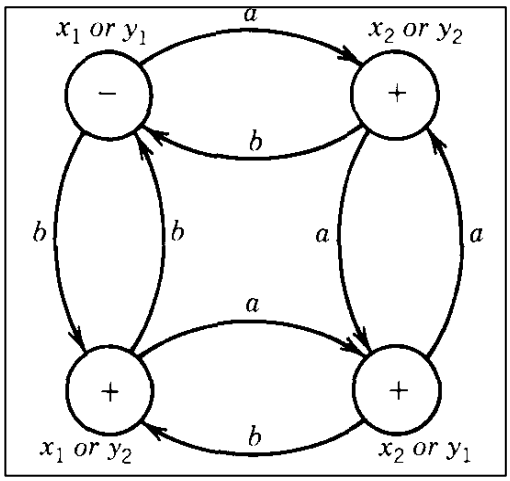
FA1 menjadi mesin di bawah ini yang menerima semua kata yang diakhiri dengan a



dan biarkan FA2 menjadi mesin di bawah ini yang menerima semua kata dengan angka ganjil huruf (panjang ganjil):



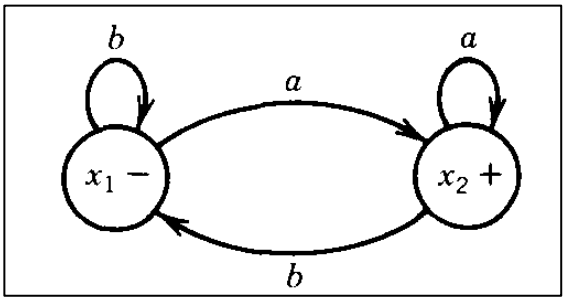
Menggunakan algoritma menghasilkan mesin di bawah ini yang menerima semua kata yang baik memiliki jumlah huruf ganjil atau yang diakhiri dengan a



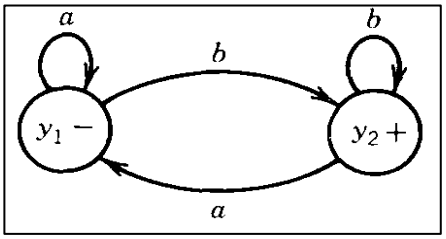
Satu-satunya state yang bukan merupakan state + adalah state -. Untuk kembali ke awal menyatakan, sebuah kata harus memiliki jumlah huruf genap dan diakhiri dengan b.

CONTOH (Dalam pembuktian Teorema)

Biarkan FA menjadi:

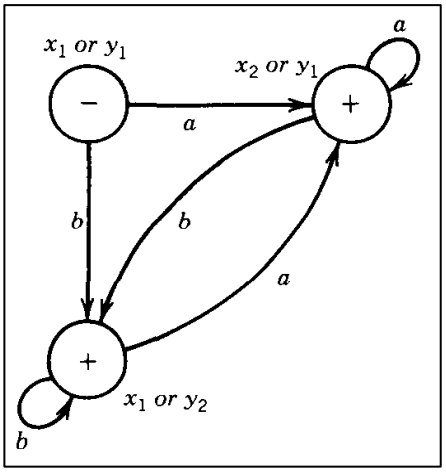


yang menerima semua kata yang berakhiran a, dan biarkan FA2 menjadi:



yang menerima semua kata yang berakhiran b.

Dengan menggunakan algoritma, menghasilkan:



yang menerima semua kata yang berakhiran a atau b, yaitu, semua kata kecuali A. Notice bahwa keadaan x2 atau Y2 tidak dapat dicapai karena x2 berarti "kita baru saja membaca" a" dan Y2 berarti "kami baru saja membaca a b.

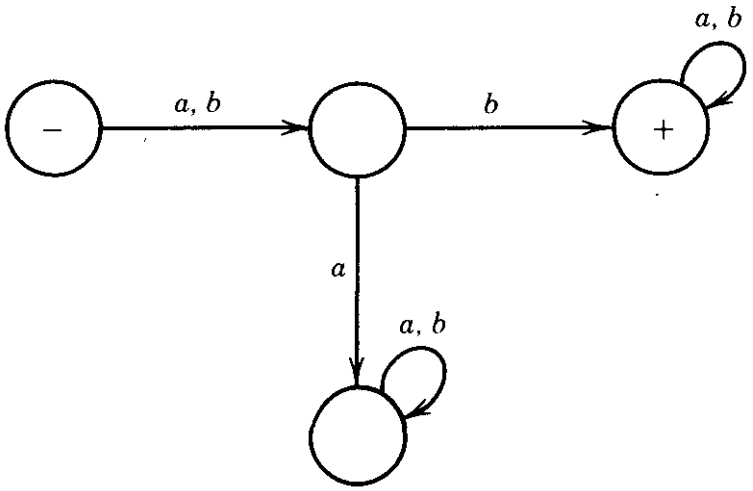
Kami masih memiliki dua aturan untuk dijalankan.

Aturan 3

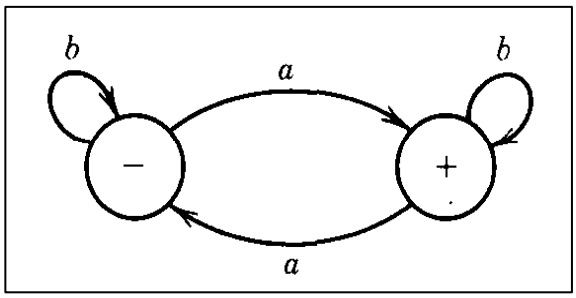
Jika ada FA I yang menerima bahasa yang ditentukan oleh ekspresi reguler r, dan FA2 yang menerima bahasa yang didefinisikan oleh regular ekspresi r 2 maka ada FA3 yang menerima bahasa yang didefinisikan oleh rangkaian r1 r2, produk bahasa produk.

Sekali lagi, kami akan memverifikasi aturan ini dengan algoritma konstruktif. Biarkan L, menjadi bahasa semua kata dengan b sebagai huruf kedua. Satu mesin

yang menerima L, adalah FA1

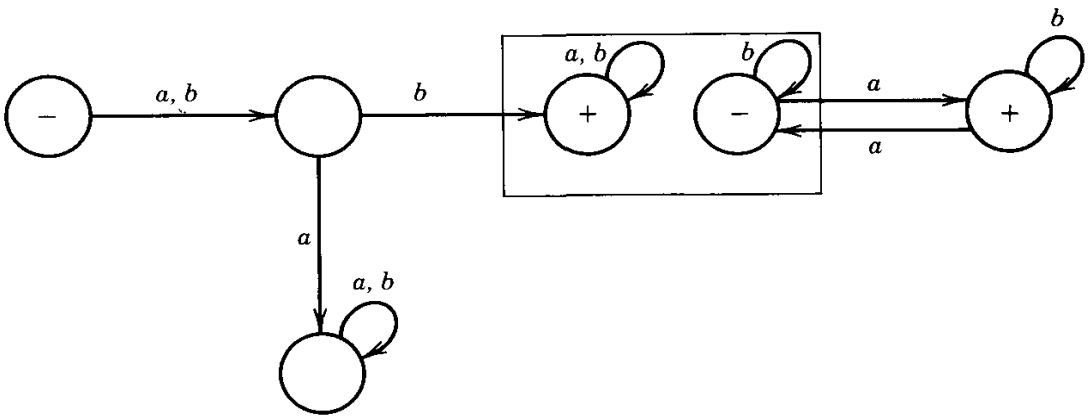


Biarkan L2 menjadi bahasa semua kata yang memiliki bilangan ganjil a. Satu mesin untuk L2 adalah FA2 di halaman berikutnya.



Sekarang perhatikan string input ababbaa. Ini adalah kata produk bahasa LIL2, karena merupakan gabungan kata dalam L, (ab) dengan kata di L2 (abba). Jika kita mulai menjalankan string ini di FAI, kita akan mencapai + menyatakan setelah huruf kedua. Jika kita sekarang bagaimana bisa melompat secara otomatis ke FA2, kita bisa mulai menjalankan apa yang tersisa dari input, abbaa, mulai di - negara. Masukan yang tersisa ini adalah sebuah kata di L2 , jadi itu akan menyelesaikannya jalur dalam state + FA2. Pada dasarnya, inilah yang ingin kami bangun-sebuah FA3 yang memproses bagian pertama dari string input seolah-olah itu adalah FAI; lalu kapan mencapai status FA1 +, berubah menjadi status - pada FA2. Dari sana itu terus memproses string hingga mencapai status + pada FA2, dan kami kemudian dapat menerima masukan.

Untuk sementara, katakanlah FA3 terlihat seperti ini:



Sayangnya, ide ini, meskipun sederhana, tidak berhasil. Kita bisa melihat ini dengan mempertimbangkan string input yang berbeda dari bahasa produk yang sama. Itu kata ababbab juga ada di L1L2, karena abab ada di L, (memiliki b sebagai yang kedua huruf) dan bab di L2 (memiliki angka ganjil dari beberapa a).

Jika kita menjalankan string input ababba terlebih dahulu di FAI, kita endapatkan state + setelahnya dua huruf, tetapi kita tidak boleh mengatakan bahwa kita sudah selesai dengan L, bagian dari masukan. Jika kita berhenti berjalan di FAI setelah ab, kita akan mencapai + di FAI, tetapi string input abbab yang tersisa tidak dapat mencapai + pada FA2 karena memiliki bilangan a genap.

Ingatlah bahwa FA1 menerima semua kata dengan jalur yang berakhir pada state akhir.Mereka bisa melewati state terakhir itu berkali-kali sebelum berakhir di sana. Ini adalah kasus dengan input abab. Mencapai + setelah dua huruf. Namun, kami harus terus menjalankan string pada FA1 untuk dua huruf lagi. Kami memutar kembali menjadi + dua kali. Kemudian kita bisa melompat ke FA2 dan menjalankan string bab yang tersisa di FA2 . Bab input akan dimulai pada FA2 dalam state - dan berakhir di State +

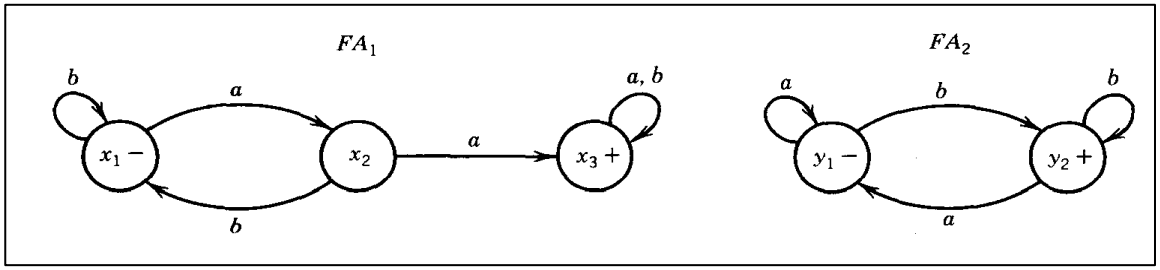
Masalah kita adalah ini: "Bagaimana kita tahu kapan harus melompat dari FA1 ke FA2?" Dengan input ababbaa kita harus melompat saat pertama kali mencapai + di FA1 . Dengan input ababbab (yang berbeda hanya pada huruf terakhir), kita harus tetap di FAI sampai kita mengulang kembali ke state + beberapa kali sebelumnya melompat ke FA2. Bagaimana bisa finite automaton, yang harus membuat transisi pada setiap huruf input tanpa melihat ke ke berikutnya untuk melihat sisa stringnya, kapan tahu harus melompat dari FA1 ke FA2?

Ini adalah poin, dan ini melibatkan beberapa ide baru. Kita harus membangun mesin yang memiliki karakteristik memulai seperti FAI dan mengikutinya sampai memasuki state akhir pada saat itu merupakan pilihan yang tercapai. Kita melanjutkan FA1 untuk mencapai + yang lain atau yang lain kami beralih ke state awal FA2 dan mulai beredar di sana. Ini adalah rumit, karena r, bagian dari string input dapat menghasilkan alphabet sesuai dengan pilihan kata (jika ada bintang di dalamnya), dan kita tidak bisa memastikan kapan harus melompat keluar dari FA1 dan menjadi FA2

Seperti sebelumnya, pertama-tama kami mengilustrasikan bagaimana membangun FA3 seperti itu untuk contoh spesifik.Dua mesin yang akan kita gunakan adalah

FAI = mesin yang hanya menerima string dengan double a di dalamnya dan,

FA2 = mesin yang menerima semua kata yang berakhiran huruf b.

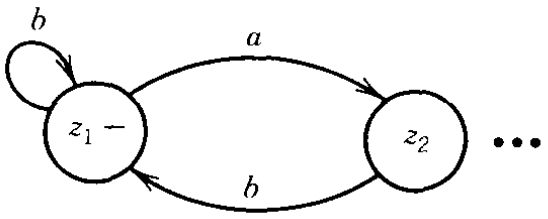


Kita akan mulai dengan keadaan z1, yang persis seperti x1. Ini adalah state awal, dan itu berarti string input sedang dijalankan di FA1 . Dari z, jika kita membaca a b, kita harus kembali ke state yang sama x1 , yaitu z1 lagi. Dari z1 jika kita membaca a, kita harus menuju ke state x2 karena kita tertarik untuk melihatnya bagian pertama dari string input adalah kata yang diterima oleh FA1 . Oleh karena itu, z2 sama dengan x2. Dari state z2 jika kita membaca a b, kita harus kembali ke zl. Oleh karena itu, kami memiliki hubungan

ZI = X1

Z2 picur X2

Gambar FA3 dimulai seperti gambar FAI.



Sekarang jika kita berada di z2 dan kita membaca a, kita harus pergi ke State baru z3 , yang dalam beberapa hal sesuai dengan State x3 di FA1 . Namun, x3 memiliki dual identitas. Entah itu berarti kita telah mencapai State akhir untuk babak pertama input sebagai kata dalam bahasa, untuk FA1 dan di situlah kita menyeberang dan jalankan sisa string input pada FA2 , atau itu hanyalah status lain yang harus dilalui string untuk mencapai State terakhirnya di FAI. Banyak string, beberapa di antaranya diterima dan beberapa di antaranya ditolak, melewati beberapa + negara bagian dalam perjalanan mereka melalui mesin apa pun.

Jika kita sekarang berada di z3 dalam kapasitasnya sebagai State akhir FAI untuk yang pertama bagian dari string input ini, kita harus mulai menjalankan sisa string input seolah-olah itu adalah input FA2 yang dimulai pada State yl. Oleh karena itu, arti lengkapnya berada di z3 adalah:

Z3 = x dan menjalankan FAI atau

Z3 = yl, dan mulai menjalankan FA2

Perhatikan kesamaan antara definisi disjungtif (salah satu/atau) ini dari z3 dan definisi disjungtif untuk keadaan z yang dihasilkan oleh algoritma yang diberikan untuk penambahan dua FA.

Jika kita berada dalam State z3 dan kita membaca a, sekarang kita memiliki tiga kemungkinan interpretasi untuk state di mana ini menempatkan kita:

- kami kembali di x3 terus menjalankan string di FA1 atau
- kami baru saja selesai di FAI dan kami sekarang di Yl mulai berjalan di FA2 atau
- kami telah mengulang dari yl kembali ke Yi saat sudah berjalan FA2

maka dengan X3 atau y, (karena berada di y, adalah sama apakah kita berada ada untuk pertama kalinya atau tidak)sama dengan z3

Oleh karena itu, jika kita berada di z3 dan kita membaca a, kita mengulang kembali ke z3. Jika kita berada dalam keadaan z3 dan kita membaca a b, kita pergi ke keadaan z4, yang memiliki berikut artinya

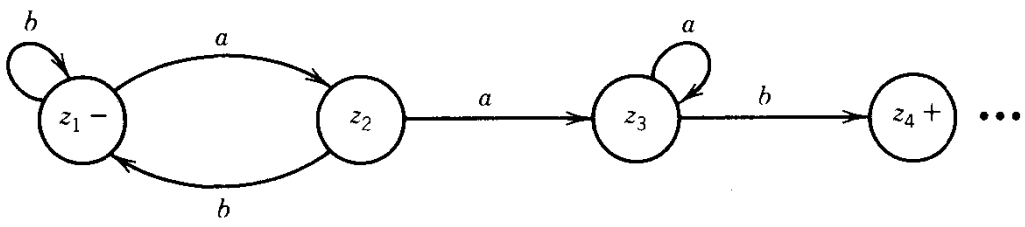
+ z4 = kami masih di x3 terus berjalan di FA1, atau

+ z4 = kita baru saja selesai menjalankan FA1 dan sekarang berada di y] pada FA2, atau

+ z4 = kita sekarang di Y2 di FA2 setelah sampai di sana melalui y,

= x3 atau y1 atau y2

telah dipecah menjadi dua bagian, yang pertama dari x, ke x3 dan detik dari y, ke y2; oleh karena itu, itu harus diterima, jadi z4 adalah keadaan akhir. Sejauh ini mesin kami terlihat seperti ini:



Jika kita berada di z4 dan kita membaca a, pilihan kita adalah:

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

tersisa di x3 dan terus berjalan di FA1, atau

baru saja menyelesaikan FA1 dan mulai dari yi, atau

setelah pindah dari Y2 kembali ke Yi di FA2

= x3 atau Y1

Namun, ini persis definisi z3 lagi. Jadi, secara ringkas, jika kita berada di z4 dan membaca a, kita kembali ke z 3.

Jika kita berada di z4 dan membaca a b, pilihan kita adalah:

tersisa di X3 dan terus berlari di FA1, atau

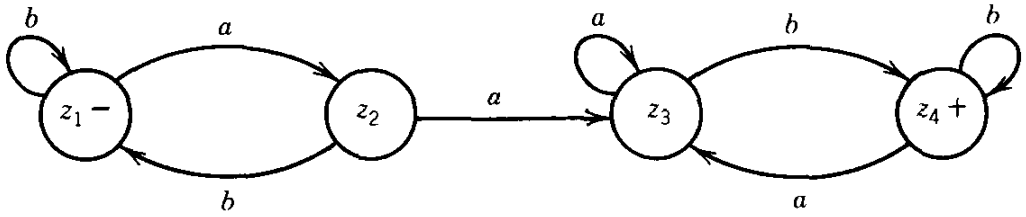
baru saja menyelesaikan FAI dan mulai dari y1,atau

setelah mengulang kembali dari Y2 ke Y2 berjalan di FA2

= x3 atau Yi atau Y2

= Z4

Dengan demikian, jika kita berada di z4 dan membaca a b, kita mengulang kembali ke z4. Seluruh mesin kemudian terlihat seperti ini:



Jadi, kami telah menghasilkan mesin yang menerima string yang sama persis memiliki bagian depan dengan double a diikuti dengan bagian belakang yang berakhiran b. Ini bisa kita lihat karena tanpa double a kita tidak akan pernah sampai ke z3 dan kita diakhiri dengan z4 hanya jika seluruh kata berakhiran b

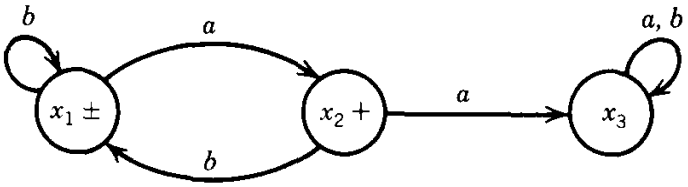
Secara umum, kita dapat menggambarkan algoritma untuk membentuk mesin FA3 sebagai mengikuti. Pertama kita membuat state z untuk setiap state x nonfinal di FA1 . Untuk setiap state akhir di FAI kami menetapkan state z yang menyatakan opsi yang kami miliki melanjutkan FAI atau memulai FA2. Dari sana kami menetapkan state z untuk semua situasi bentuk

ada di X sesuatu melanjutkan FA1, atau, baru saja mulai yi akan melanjutkan FA2, atau ada di Ysesuatu melanjutkan FA2

Jelas hanya ada banyak kemungkinan untuk keadaan z seperti itu, jadi FA3 adalah mesin yang terbatas. Transisi dari satu keadaan z ke keadaan lain untuk setiap huruf alfabet ditentukan secara unik oleh aturan transisi di FA, dan FA2. Jadi FA3 adalah otomat terbatas yang terdefinisi dengan baik yang dengan jelas melakukan apa yang kita inginkan, yaitu, ia hanya menerima string yang pertama kali mencapai status akhir pada FA1 dan kemudian mencapai status akhir di FA2.

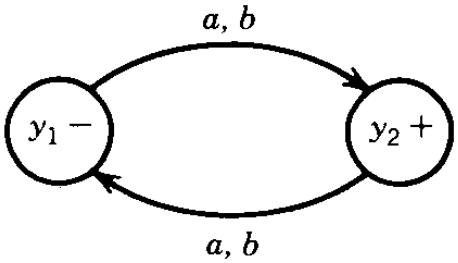
CONTOH (Di dalam bukti Teorema 6)

Misalkan FA1 menjadi

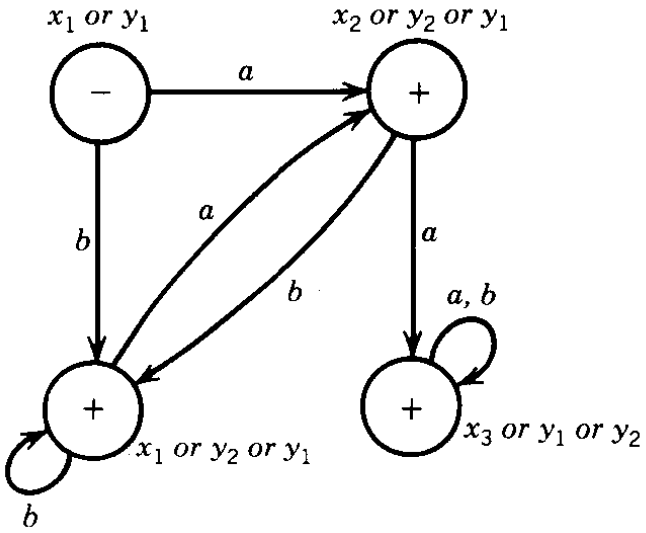


yang menerima bahasa L, dari semua kata yang tidak mengandung substring aa.

FA2



yang menerima bahasa L 2 dari semua kata dengan jumlah huruf ganjil. Menggunakan algoritma di atas, kami memproduksi mesin berikut untuk menerima: bahasa produk L L2



Semua state kecuali state - adalah state akhir. state - dibiarkan seketika surat input dibaca, dan tidak pernah bisa dimasukkan kembali. Oleh karena itu, bahasa mesin ini menerima semua kata kecuali Ɛ. Ini sebenarnya adalah bahasa produk L1 L2, karena jika sebuah kata w memiliki jumlah huruf ganjil, kita dapat memfaktorkannya sebagai (Ɛ)(w), di mana Ɛ di L, dan w di L2 . Jika w memiliki bilangan genap (bukan 0) huruf, kita dapat memfaktorkannya sebagai

w = (huruf pertama) (sisanya)

dimana (huruf pertama) harus di L, dan (selebihnya) di L2. Hanya kata Ɛ tidak dapat difaktorkan menjadi bagian di L1 dan bagian di L2.

Kami sekarang siap untuk aturan terakhir.

Aturan 4

Jika r adalah ekspresi reguler dan FA1 adalah finite Automata yang menerima persis bahasa yang didefinisikan oleh r, maka ada FA yang disebut FA2 yang akan menerima persis bahasa yang didefinisikan oleh r*.

Bahasa yang didefinisikan oleh r* harus selalu berisi kata nol. Menerima string nol A kita harus menunjukkan bahwa keadaan awal juga merupakan keadaan akhir. Ini bisa menjadi perubahan penting di mesin FA1 , karena string yang kembali ke x, mungkin tidak diterima sebelumnya. Mereka mungkin tidak dalam bahasa dari ekspresi r. Pembangunan mesin baru kami harus dilakukan dengan hati-hati.

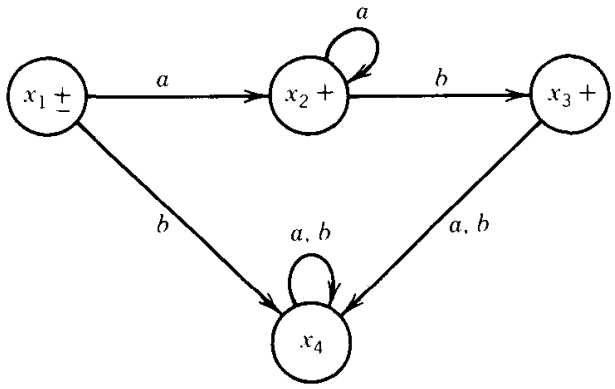
Mari kita perhatikan ekspresi reguler

$$r = a^* + aa^*b$$

Bahasa yang didefinisikan oleh r adalah semua string hanya a dan string beberapa

(bukan nol) a berakhiran b tunggal. Penutupan bahasa ini didefinisikan oleh (a* + aa*b)*, yang mencakup semua kata yang setiap b memiliki a pada kiri. Di sini r* jelas tidak sama dengan r, karena kata-kata seperti aba dan ababaaa dalam r* tetapi tidak dalam bahasa r.

Mesin yang kami gunakan untuk menerima r adalah FAI yang digambarkan di bawah ini.



Perhatikan bahwa x4 adalah status penolakan. Setiap string yang masuk akan tetap di sana dan adalah akhirnya ditolak. Sebuah kata yang pergi ke x2 dan berhenti di sana adalah kata dari semua a dan diterima. Untuk sampai ke x3 dan berhenti di sana, kita membutuhkan tepat satu b setelah a. Memang benar bahwa x, juga merupakan keadaan akhir, tetapi satu-satunya kata yang berakhir ada Ɛ

Mesin yang akan kita buat, FA2, untuk menerima bahasa yang didefinisikan oleh r* dimulai sebagai berikut.

$$\pm \; z_1 = x_1$$

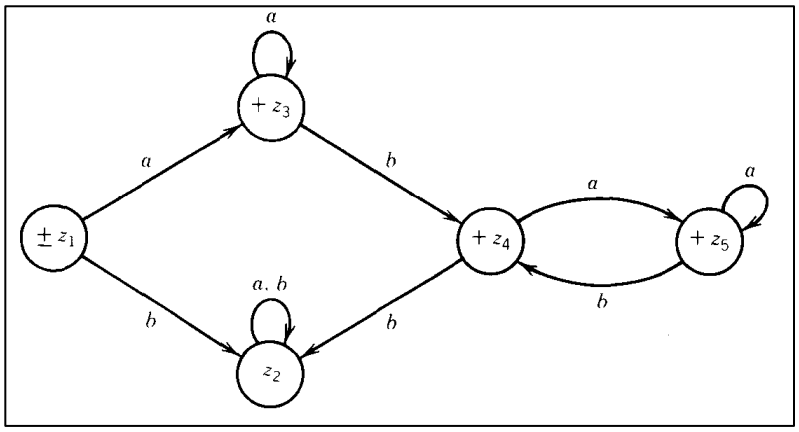
Jika kita berada di z1 dan membaca a b, kita menuju ke keadaan tolak x4, yang kita sebut z2.

$$Z2 = X4$$

Jika kita berada di z, dan membaca a, kita pergi ke z3, yang berarti sedikit lebih dari x2 sendirian

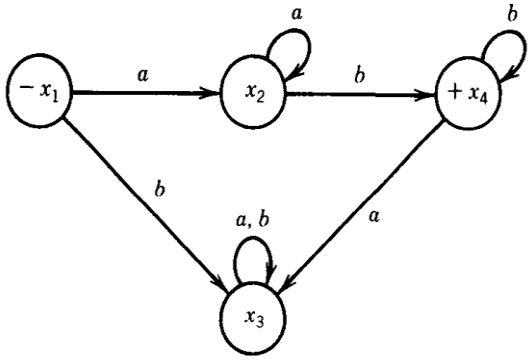
- ✓ +z3 = X2 dan kami terus memproses bagian tengah dari faktor yang lebih panjang dari tipe r yang mungkin hanya salah satu dari banyak substring dari yang terdiri dari kata input kata atau
- ✓ +z3 = X2 kami baru saja menerima bagian dari string input sebagai dalam bentuk yang tepat untuk r dan sekarang kita kembali ke x, mulai lagi di bagian berikutnya dari string input

Ini menyelesaikan deskripsi seluruh mesin. Hal ini digambarkan di bawah ini.



(a* + aa*b)*

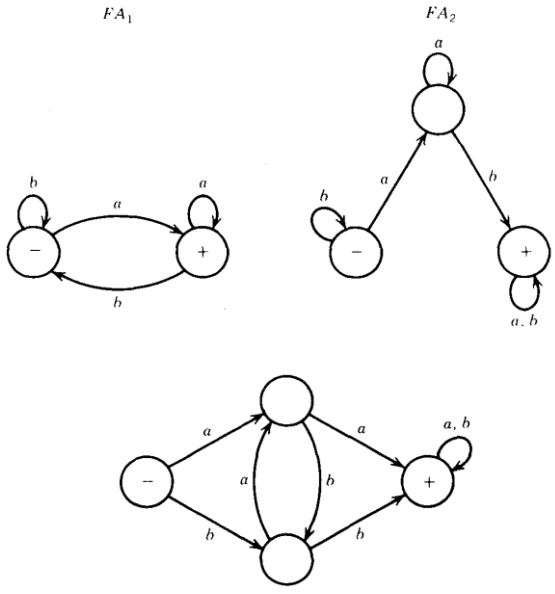
Satu FA yang menerima bahasa ini adalah:



r* = (aa*bb*)*:

5.9 Soal Latihan

1. Pertimbangkan automata berikut:



Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

Temukan ekspresi reguler r1 , r 2, r3 masing-masing, dan sederhanakan ekspresi jika memungkinkan. Jelaskan bahasa-bahasa ini dalam bahasa Inggris. Menggunakan algoritma dalam bab ini dan tiga FA dari Soal 1 temukan FA untuk hal-hal berikut:

a. r1 + r2
b. r1 + r 3
c. r 2 + r 3
b. r1 r 2
c. r1r 3
d. r2r3
e. r2rl
f. r1r1,
g. (r1)*
h. (r 2)*
i. (r 3)* direalisasi!

Bab 6

Nondeterministik Finite Automata (NFA)

6.1 Nondeterministik Finite Automata (NFA)

Pada bab yang terdahulu membahas FA yang sederhana, Finite automata sebagai mesin acceptor yang berhingga karena hanya memiliki himpunan state berhingga disebut DFA. DFA memproses string dan menerima atau menolaknya, DFA. kata sifat " deterministik " menandakan bahwa mesin automata memiliki satu dan hanya satu pilihan kapan sajatransisi menuju berikutnya. Dalam situasi tertentu, kami akan memperkenalkan automata nondeterministik, atau NFA. NFA dapat memiliki beberapa opsi dan karena itu dapat muncul untuk punya pilihan yaitu transisi menuju lebih dari satu state berikutnya. Di dunia deterministik, atau setidaknya di dunia deterministik dunia komputer, apa peran nondeterminisme? Saat kita terkadang mengatakan bahwa NFA dapat membuat pilihan terbaik. Cara yang lebih baik untuk memvisualisasikan nondeterminisme adalah dengan berpikir bahwa NFA mengeksplorasi semua pilihan dan tidak membuat keputusan sampai semua pilihan telah dianalisis. utama alasan untuk memperkenalkan nondeterminisme adalah karena menyederhanakan solusi dari banyak masalah, supaya hal ini sederhana maka NFA akan kita definisikan,

NFA dapat didefinisikan kumpulan dari tiga hal:

a. Himpunan state berhingga dengan satu state awal (-) dan beberapa state akhir (+).
b. Alfabet sebagai huruf yang diinputkan.
c. Himpunan transisi state yang menjelaskan bagaimana melanjutkan dari setiap state ke state yang lain di diberi input dengan huruf alfabet (tetapi tidak ada null string), di mana kemungkinan lebih dari satu input alphabet menuju state berikutnya

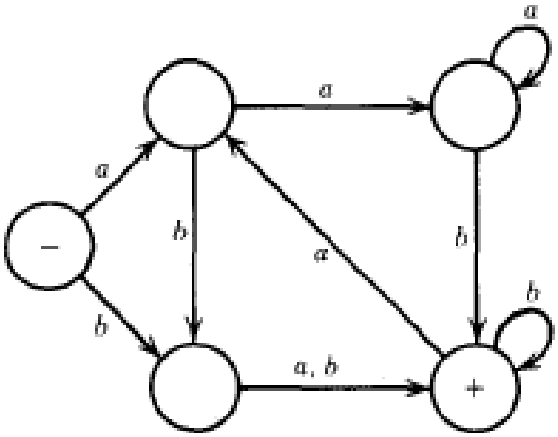
Setiap FA akan memenuhi definisi NFA.sehinga;

a. Setiap FA adalah NFA.
b. Setiap NFA equevalen dengan TG
c. Dengan teorema Kleene, setiap TG equevalen dengan FA.

Karena itu bahasa yang diterima oleh FA adalah bahasa yang yang didalamnya diterima oleh NFA demikian pula oleh TG, sehingga bisa dapat dikatakan bahwa FA = NFA

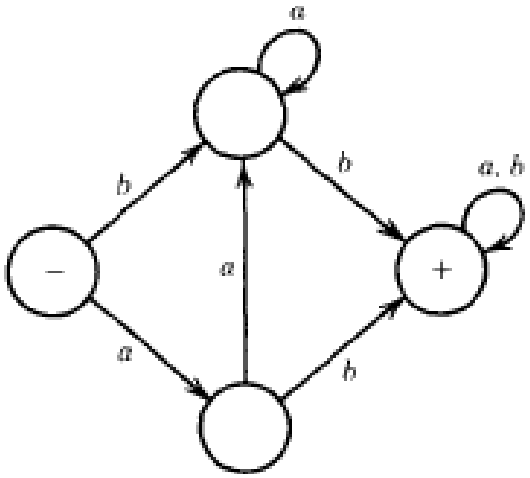
Jangan sampai kita salah mengartikan persamaan FA = NFA yang berarti bahwa setiap NFA itu sendiri adalah FA. Ini tidak benar. Hanya saja untuk setiap NFA ada beberapa FA yang setara dengannya sebagai penerima bahasa. NFA terkadang lebih mudah atau lebih intuitif untuk digunakan daripada definisi yang diterima oleh FA. Salah satu contohnya adalah mesin untuk menggabungkan dua FA, satu yang menerima bahasa didefinisikan dengan ekspresi reguler r1, dan yang lainnya untuk bahasa didefinisikan dengan ekspresi reguler r2. Jika awal menyatakan dalam dua mesin ini tidak memiliki edge/sisi yang masuk ke dalamnya, kami dapat menghasilkan NFA3 yang menerima persis bahasa r1, + r2 dengan menggabungkan dua state awal. Sebagaimana diilustrasikan FA di bawah ini. (Levin 2009)

FA 1



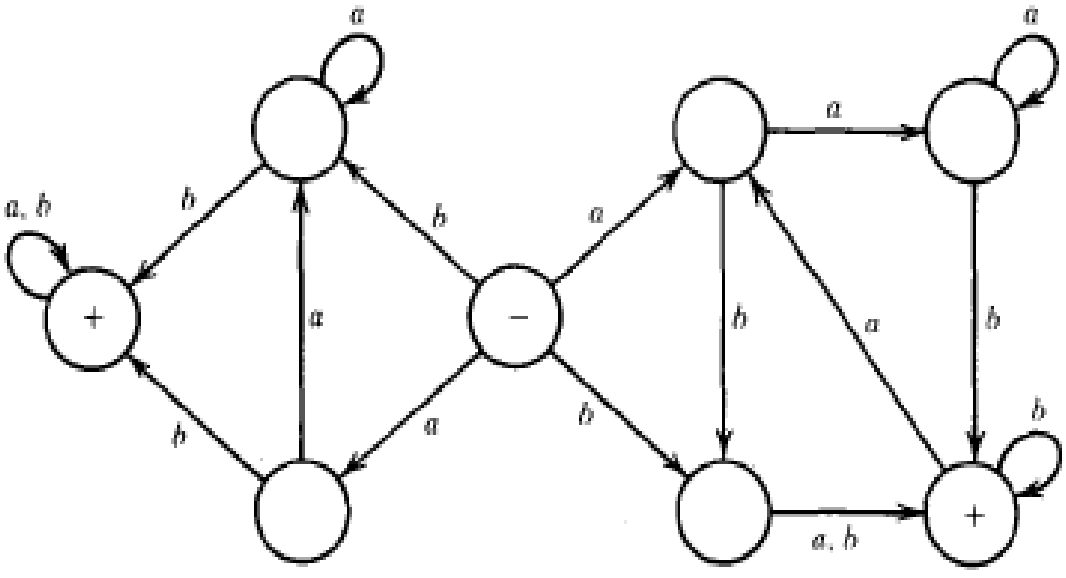
Gambar 6.1a: NFA1 a.

FA2



Gambar 6.1b: NFA2

NFA3 = FA1+FA2



Gambar 6.1c: NFA1 + NFA2

Definisi Non Deterministik Finite Automata

Kita juga dapat mendefinikan NFA penerima bahasa dengan model yang lain, yaitu bahwa NFA didefinisikan mesin penerima bahasa disebut sebagai mesin,

$M = (Q,\Sigma, \delta, q0, F)$

Q adalah himpunan state terbatas

Σ input alphabet yang terbatas

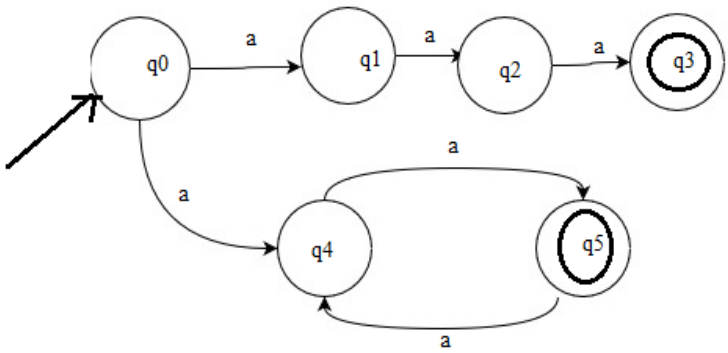
$\delta : Q \times (\Sigma \cup \{ø\}) \rightarrow 2Q$. fungsi transisi

$s \in Q$ disebut initial state/state awal/state penerimaan

$F \subseteq Q$ final states/ state penerimaan/state akhir

Perhatikan bahwa ada tiga perbedaan utama antara definisi ini dan definisi DFA. Dalam NFA menerima bahasa, adalah di power set 2Q, sehingga nilainya bukan elemen tunggal Q, Subset ini mendefinisikan himpunan semua kemungkinan status yang dapat dicapai oleh transisi. Jika, misalnya, keadaan saat ini adalah q1, simbol a adalah input alphabetnya maka (q1, a) = {q0, q2}

Sebagai contoh graph transisnya sbb,



Gambar 6.2: NFA

Sehingga perbedaan yang sangat sederhana adalah dapat dilihat dasi table transisinya sebagaimana table sebagai berikut

Old state	Input a
Q0	Q1,Q4
Q1	Q2
Q2	Q3
Q3	ø
Q4	Q5
Q5	Q4

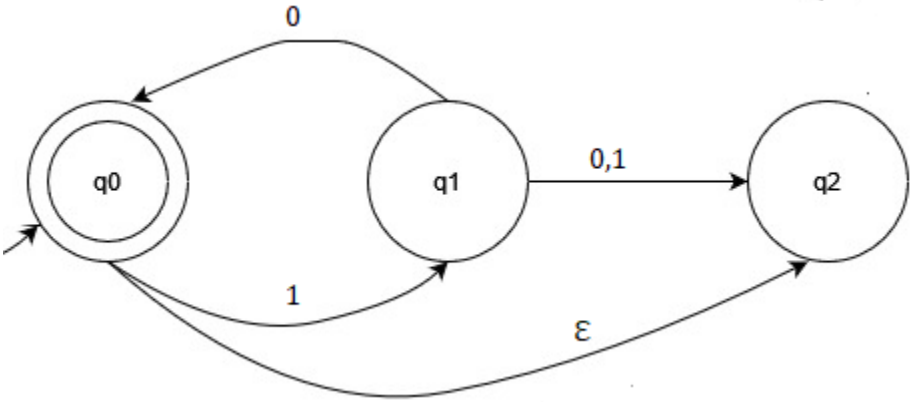
Definisi

Bahasa L yang diterima oleh nfa $M = (Q,\Sigma, \delta, q0, F)$ didefinisikan sebagai set semua string yang diterima dalam pengertian di atas. Secara formal, bahwa

$$L(M) = \{w : (q0, w) F = \} .$$

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

Dengan kata lain, bahasa terdiri dari semua string w yang ada menuju sisi berlabel w dari state awal transisi ke beberapa state akhir.



Gambar 6.3: NFA

Bahasa apa yang diterima oleh mesin pada Gambar diatas, maka untuk melihat dari graph diatas bahwa satu-satunya cara NFA menerima barisan string yang dapat diterima berhenti pada state q0. Oleh karena itu, menerima bahasa yang diterima dapat ditulis dengan notasi

$$L = \{(10)n : n \geq 0\}$$

Apa yang terjadi ketika mesin ini diberikan input w berupa string w = 110? Setelah mesin NFA membaca awalan 11, mesin bergerak dirinya menuju state q2, dengan demikian maka mesin tidak bergerak lagi karena tidak ada inputan yang menuju state dengan demikian dengan inputan dari tidak terdefinisikan karena

$$\delta(q0,1) = q1$$

$$\delta(q1,) = q2,$$

sedangkan tidak ada lagi $\delta(q2, ?) = ?$

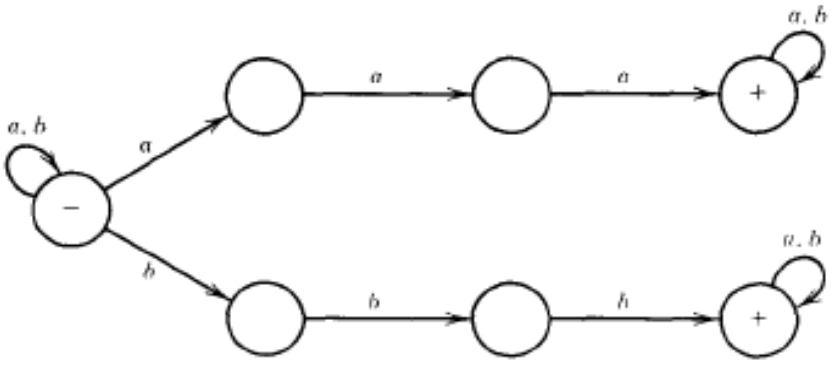
dengan demikian dikatakan bahwa mesin NFA tidak dapat bergerak lagi atau dapat dikatakan sebagai konfigurasi yang mati, dapat dinotasikan sebagai berikut,

$$\delta* (q0, 110) = \emptyset.$$

Jadi, mesin tidak dapat mencapai state akhir, dalam hal ini saat proses membaca w = 110, dan karenanya itu string tidak diterima.

Contoh 6.1.

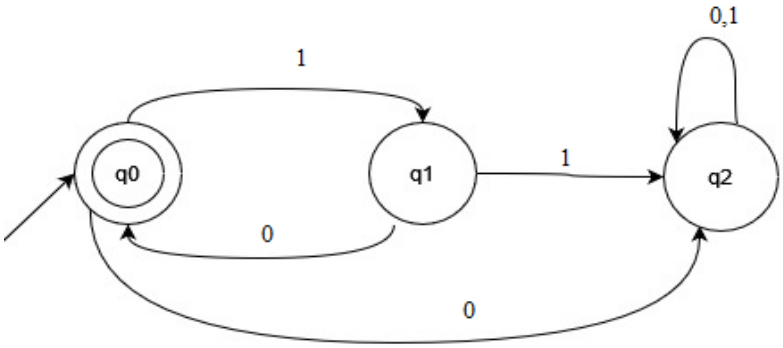
Untuk contoh, bahasa semua string yang mengandung triple a (substring aaa) atau triple b (substring bbb) diterima oleh mesin NFA



Gambar 6.4: NFA menerima minimal triple a dan triple b

Contoh 6.2.

Sebuah NFA ditunjukkan pada Gambar Ini



Gambar 6.5: Contoh Mesin NFA

M = (Q,Σ, δ, q0, F)

Q = { q0, q1, q2)}) adalah himpunan state terbatas

Σ {0,1}) input alphabet yang terbatas

$\delta : Q \times (\Sigma \cup \{\emptyset\}) \rightarrow 2Q$. fungsi transisi

$q0 \in Q$ disebut initial state/state awal/

$q0 \subseteq Q$ state akhir

Apakah string , 1010, dan 101010, diterima oleh NFA, maka untuk membuktikan ini perlu diuraikan sebagaimana berikut,

1) String 1010

Bukti

δ(q0, 1)= q1

δ(q1, 0)= q0

δ(q0, 1)= q1

δ(q1, 1)= q2,

karena q2 adalah state akhir maka string 1010 diterima oleh mesin NFA, atao dengan kata lain bahwa 1010 elemen dari L(M)

2) String 101010

Bukti

δ(q0, 1)= q1

δ(q1, 0)= q0

δ(q0, 1)= q1

δ(q1, 0)= q0

δ(q0, 1)= q1

δ(q1, 0)= q2

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

karena q2 adalah state akhir maka string 101010 diterima oleh mesin NFA, atau dengan kata lain bahwa 101010 elemen dari L(M).

6.2 EQUIVALENCE DFA dan NFA

Kita sekarang sampai pada pertanyaan mendasar. Dalam pengertian apa itu DFA dan NFA apa yang berbeda? Jelas, ada perbedaan dalam definisi mereka, sebagiaman di jelaskan pada bab terdahulu, tetapi ini tidak tidak berarti bahwa ada perbedaan esensial di antara mereka. Untuk mengeksplorasi pertanyaan ini, kami memperkenalkan konsep kesetaraan antara automata DFA dan NFA.

Definisi 6.2.

Dua Msin penerima, M1 dan M2, dikatakan ekuivalen jika,

$$L(M1) = L(M2)$$

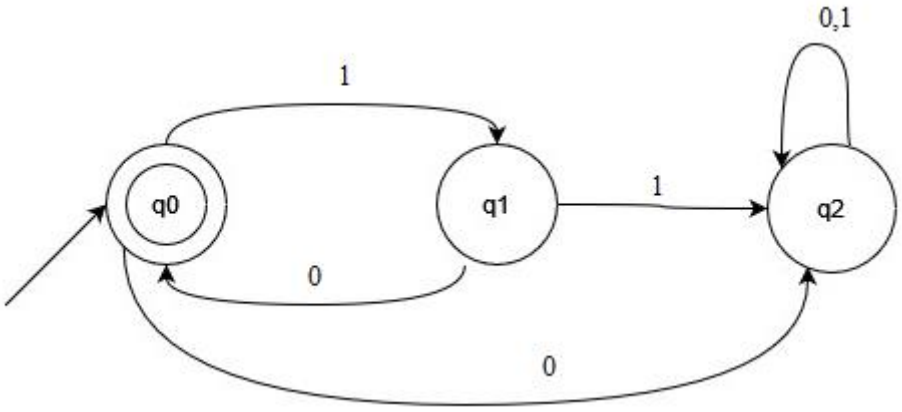
jika mereka berdua menerima bahasa yang sama.

Seperti disebutkan, umumnya ada banyak Mesin penerima bahasa tertentu, jadi setiap DFA atau NFA memiliki pengertian menerima bahasa yang setara.

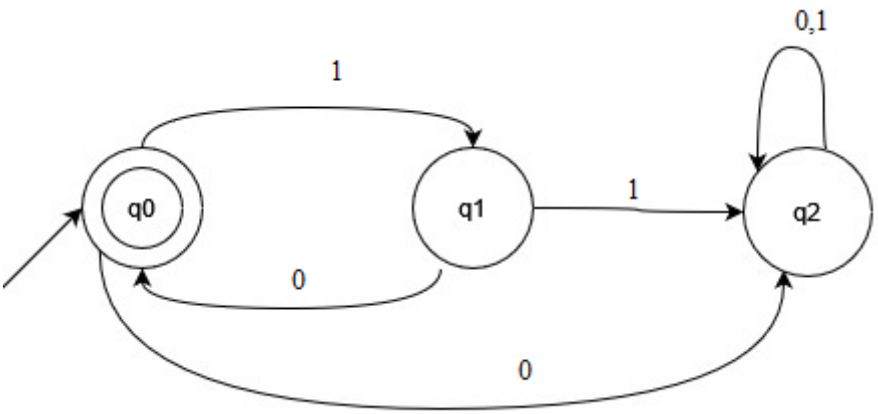
Contoh

DFA yang ditunjukkan pada Gambar dibawah ini setara dengan pada Gambar 5.5a dan 5.5b.

karena mereka berdua menerima bahasa $\{(10)n : n \geq 0\}$.



Gambar 6.6a: DFA



Gambar 6.6b: NFA

Ketika kita membandingkan type Mesin automata yang berbeda, pertanyaannya selalu muncul apakah satu type lebih baik dari yang lain. Dengan "lebih baik" yang dimaksud adalah outomata dari satu jenis dapat mencapai sesuatu yang tidak dapat dilakukan oleh mesin jenis lain. Mari kita lihat ini pertanyaan untuk DFA. Karena DFA pada dasarnya adalah jenis mesin yang terbatas dibandingkan nfa, jelas bahwa bahasa apa pun yang diterima oleh dfa juga diterima oleh beberapa NFA. Tetapi kebalikannya tidak begitu jelas. Kami telah menambahkan nondeterminisme, jadi setidaknya bisa dibayangkan ada bahasa yang diterima oleh beberapa NFA yang mungkin tidak dapat menemukan di DFA. Tapi ternyata tidak begitu. Type DFA dan NFA sama baiknya: Untuk setiap bahasa yang diterima oleh beberapa NFA ada DFA yang menerima bahasa yang sama. Hasil ini tidak jelas dan tentu harus dibuktikan. Argumennya, kita ilustrasikan dengan contoh sederhana.

CONTOH 5.3. gambar 5.6a dan gambar 5.6b equivalen

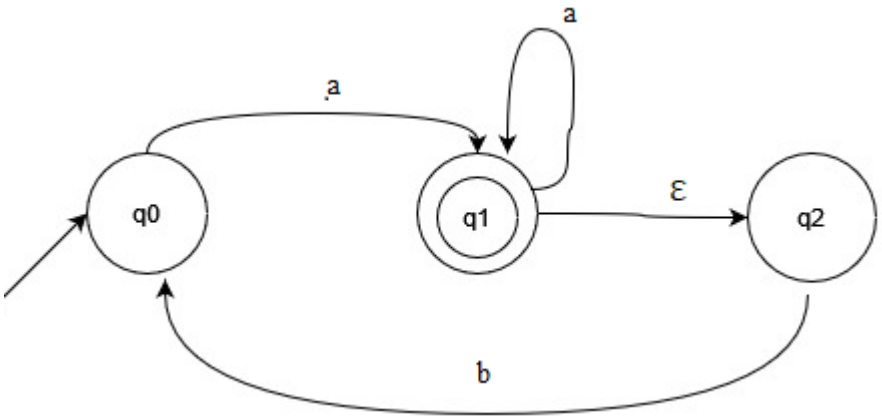
Ubah NFA pada Gambar ini menjadi DFA yang setara. NFA dimulai dari state q0, sehingga state awal dfa akan diberi label {q0}. Setelah membaca a, NFA dapat meraih q1 atau, dengan membuat - transisi, menuju state q2. Oleh karena itu, dfa yang sesuai harus memiliki state berlabel {q1, q2} dan transisinya adalah

$$\delta(\{q0\}, a) = \{q1, q2\}$$

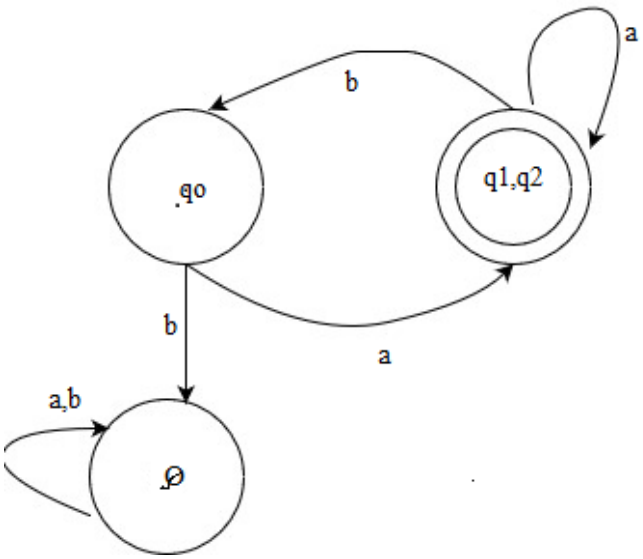
Dalam state q0, NFA tidak memiliki transisi yang ditentukan ketika inputnya adalah b; karena itu,

$$\delta(\{q0\}, b) = \emptyset.$$

Sebuah stete berlabel ø merupakan langkah yang mustahil untuk NFA dan, oleh karena itu, berarti tidak menerima string, keadaan ini di DFA harus berupa state jebakan nonfinal.



Gambar 6.7a: DFA



Gambar 6.8b: NFA

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

Kami sekarang telah merancang state baru pada mesin DFA yaitu State {q1, q2}, Ingatlah bahwa state ini untuk DFA sesuai dengan dua kemungkinan state NFA, jadi kita harus merujuk kembali ke NFA. Jika NFA dalam keadaan q1 dan membaca a, itu bisa pergi ke q1. Selanjutnya, dari q1 nfa dapat membuat - transisi ke q2. Jika, untuk input yang sama, nfa dalam state q2, maka tidak ada transisi yang ditentukan. Karena itu

$$\delta(\{q1, q2\}, a) = \{q1, q2\}$$

Demikian pula,

$$\delta(\{q1, q2\}, b) = \{q0\}.$$

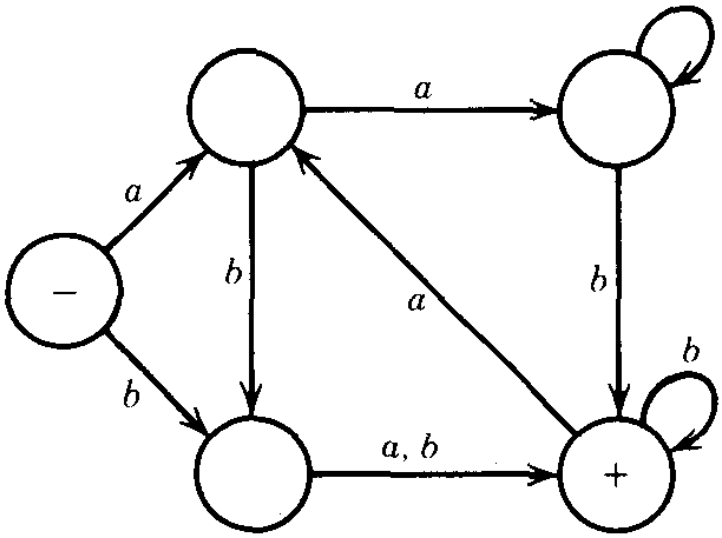
Setiap FA akan memenuhi definisi NFA.

1. Setiap FA adalah NFA.
2. Setiap NFA memiliki TG yang setara.
3. Dengan teorema Kleene, setiap TG memiliki FA yang setara.

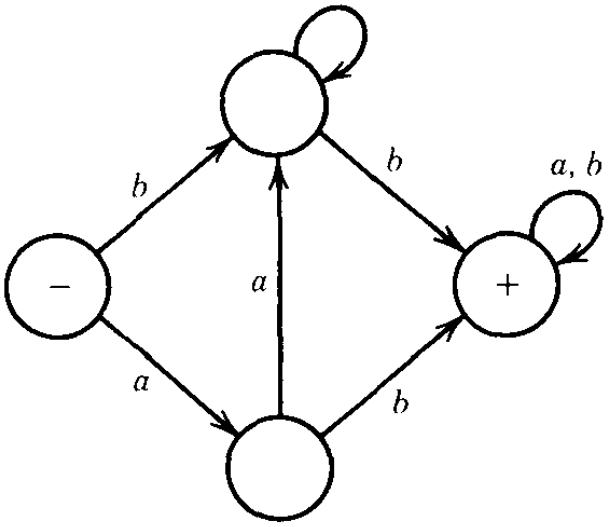
TEOREMA

$$FA = NFA$$

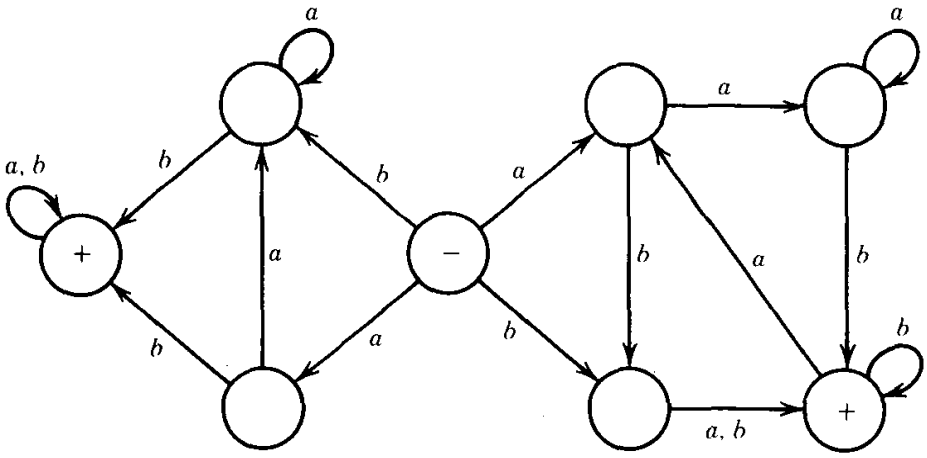
yang kami maksudkan bahwa bahasa apa pun yang dapat didefinisikan oleh NFA juga dapat didefinisikan oleh DFA dan dan sebaliknya. Kami kemudian mengatakan bahwa mereka memiliki kekuatan yang sama. Aritinya bahwa tidak sama persamaan FA = NFA yang berarti bahwa setiap NFA itu sendiri adalah FA. Ini tidak benar. Hanya saja untuk setiap NFA ada beberapa FA yang setara dengannya sebagai penerima bahasa. NFA terkadang lebih mudah atau lebih intuitif untuk digunakan daripada definisi FA bahasa yang diberikan. Salah satu contohnya adalah mesin untuk menggabungkan dua FA, satu yang menerima bahasa ekspresi reguler r, dan yang lainnya untuk bahasa ekspresi reguler r 2. Jika awal menyatakan dalam dua ini mesin tidak memiliki tepi yang masuk ke dalamnya, kami dapat menghasilkan NFA3 yang menerima persis bahasa r, + r2 dengan menggabungkan dua status awal. Ini diilustrasikan di bawah ini. Kita perhatikan FA1



FA 2, adalah

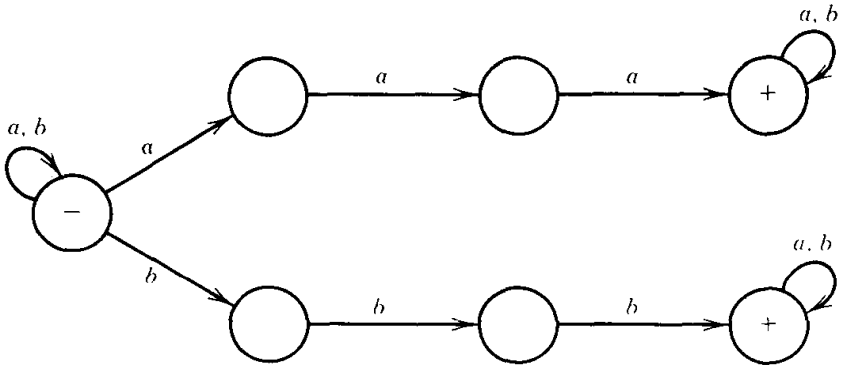


Maka NFA3 = FAl + FA2 is



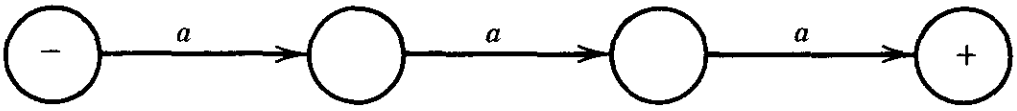
Contoh

Terkadang lebih mudah untuk memahami apa itu bahasa dari gambar NFA yang menerimanya daripada dari gambar FA. Mari kita ambil, untuk contoh, bahasa semua kata yang mengandung triple a (substring aaa) atau triple b (substring bbb) atau keduanya. NFA di bawah ini jelas tugas menerima bahasa ini.

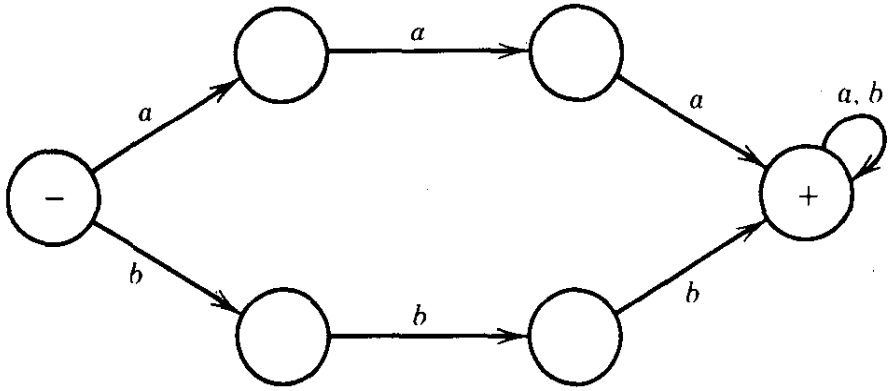


Jika sebuah kata mengandung aaa di suatu tempat di dalamnya, itu dapat diterima oleh sebuah jalan sepanjang jalan raya. Jika mengandung bbb, dapat diterima di sepanjang jalan rendah. Jika sebuah kata memiliki keduanya, kata tersebut dapat menggunakan salah satu rute. Jelas, apa pun yang mencapai + memiliki salah satu substring ini.

Sekarang mari kita bangun FA yang menerima bahasa ini. Itu harus dimulai state. Dari state awal, ia harus memiliki jalur tiga sisi untuk menerima kata aaa. Dibutuhkan tiga sisi karena jika tidak, string a yang lebih pendek dapat diterima. Oleh karena itu, kami memulai mesin kami dengan



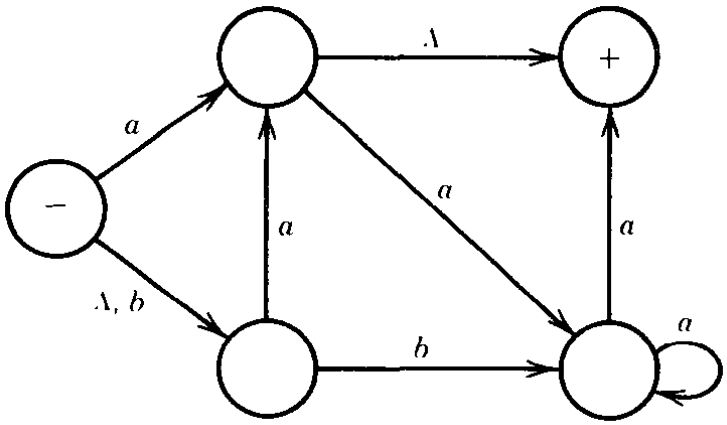
Untuk alasan yang sama, kita dapat menyimpulkan bahwa pasti ada jalan untuk bbb itu tidak memiliki loop dan menggunakan state yang sama sekali berbeda. Jika itu berbagi hal yang sama menyatakan, kita bisa mulai dengan a dan melanjutkan dengan b. Jadi kita harus memiliki setidaknya dua state lagi karena kami dapat berbagi state terakhir dari tiga sisi jalan dengan salah satu state.



Jika kita mengikuti membaca beberapa a dan membaca a b sebelum a ketiga, kita melompat ke awal jalur bbb..

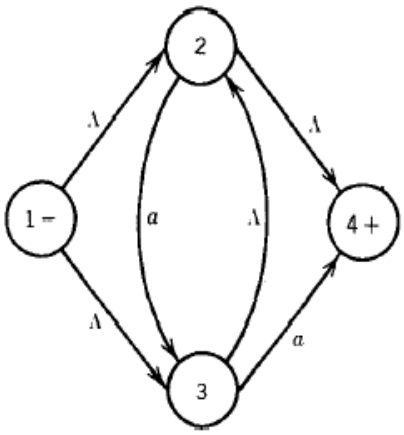
6.3 Latihan Soal

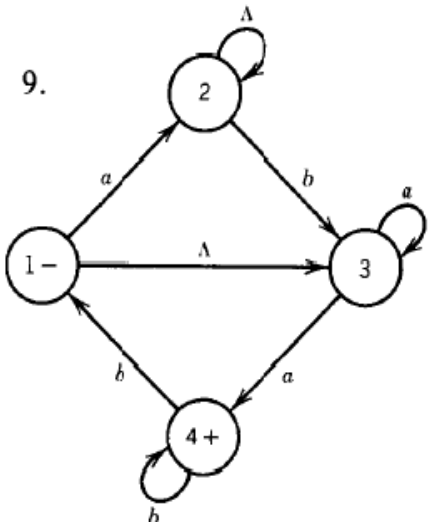
1. 1 NFA ;



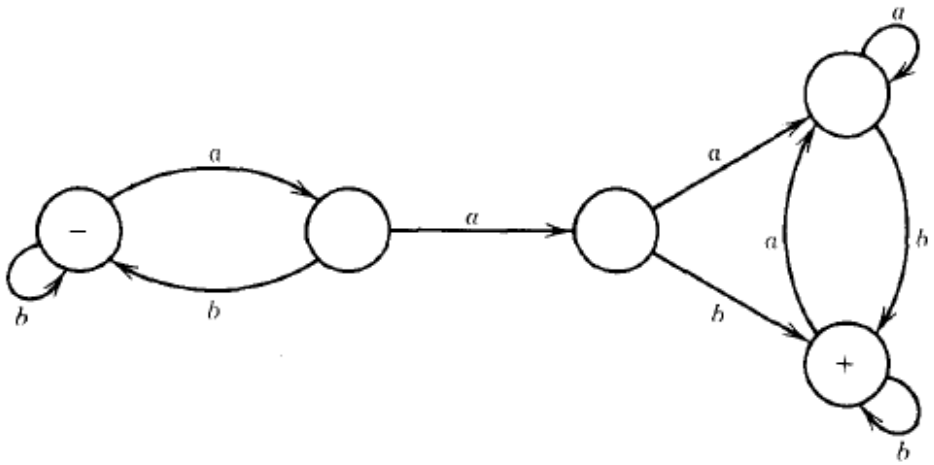
   a. Kata-kata apa yang diterima oleh mesin yang digambarkan di atas?
   b. Buktikan bahwa NFA-Ɛ memiliki yang sama dengan FA, yaitu, tunjukkan bahwa bahasa apa pun yang diterima oleh mesin dari satu jenis juga dapat diterima oleh mesin yang lain.
   c. Tunjukkan bahwa NFA-Ɛ dapat ditambahkan, digabungkan, dan dibintangi dengan cukup mudah (tanpa kondisi tambahan yang diperlukan untuk NFA).
   d. Tuliskan L (M)nya

2. Ubahlah NFA-Ɛ berikut menjadi FA.





3. Untuk bahasa yang diterima oleh mesin di bawah ini, temukan FA yang berbeda dengan empat state. Temukan NFA yang menerima bahasa yang sama dan memiliki hanya tujuh sisi.



Bab 7

Finite Automata with Output

7.1 Finite Automata with Output

Dalam diskusi ini tentang finite automata dalam bentuk model matematika komputer. Kami mengatakan bahwa string input mewakili program dan input data. Membaca huruf string bentuk analog dengan mengeksekusi instruksi yang mengubah state mesin, yaitu, mengubah isi memori, mengubah bagian kontrol dari komputer, dan sebagainya. Pada bagian ini beda dengan finite automata sebelumnya, yaitu ketika state membaca string di sisi kemudian menuju state brikutnya akan mencetak atau mengahasilkan output string. Jika kita berasumsi bahwa semua hasil output harus dilakukan pada akhir program yang berjalan, pada saat kita memiliki instruksi yang membuang buffer, maka kita memiliki jumlah karakter maksimum yang dapat program cetak, yaitu ukuran buffer. Namun, secara teoritis kita harus bisa memiliki output dengan panjang yang terbatas. Misalnya, kita mungkin hanya ingin cetak salinan input string, yang bisa panjangnya bebas. Ini adalah pertanyaan yang harus dihadapi jika kita ingin mengklaim bahwa model matematika FA dan TG mewakili mesin fisik yang sebenarnya.

Sejauh ini kita hanya menggunakan finite automata sebagai mesin penerima atau pengenal bahasa; dalam bab ini kita akan menyelidiki dua model yang berbeda untuk FA dengan kemampuan output. Ini diciptakan oleh G. H. Mealy (1955) dan, secara independen, oleh E. F. Moore (1956). Tujuan awal penemu adalah untuk merancang model matematika untuk rangkaian mesin sekuensial, yang hanya satu komponen dari keseluruhan komputer. Ini adalah komponen penting, dan seperti yang kita akan melihat, bertindak sebagai mesin dengan sendirinya. Kami akan menyajikan dua model mesin FA, dan membuktikan bahwa mereka setara, dan memberikan beberapa contoh bagaimana mereka beroperasi.

7.2 Mesin Moore

Definisi

Mesin Moore adalah kumpulan dari lima hal:

a. Suatu himpunan State berhingga qo, q1 , q2 . . . di mana qo ditunjuk sebagai state awal.
b. Alfabet huruf untuk membentuk string input Σ _= {a, b,c. .... }
c. Alfabet dari karakter output y = {x, y, x . ……..... }.
d. Tabel transisi yang menunjukkan untuk setiap state dan setiap huruf input state mencapai sate selanjutnya.
e. Tabel output yang menunjukkan karakter yang dicetak oleh setiap state yang di inputkan

Contoh 7.1.

Perhatikan contoh yang didefinisikan oleh sebuah tabel:

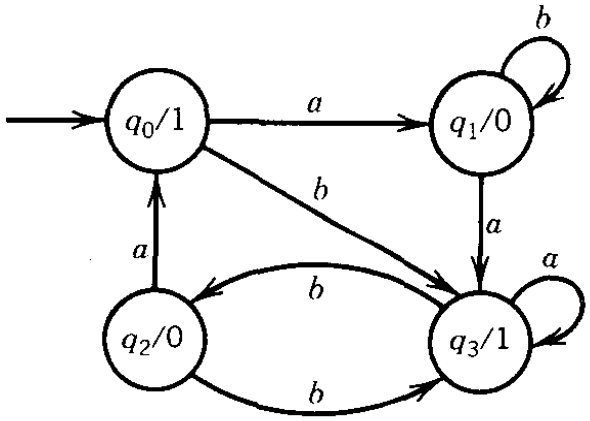
Input alfabet: Σ = {a, b}

Alfabet keluaran: Y = {0, 1 }

State: qo, q1 , q2 , q3 , (qo = status awal)

	Transition Table		Output Table
	New State		
Old State	After Input *a*	After Input *b*	(The Character Printed in the Old State)
− q_0	q_1	q_3	1
q_1	q_3	q_1	0
q_2	q_0	q_3	0
q_3	q_3	q_2	1

Representasi bergambar dari mesin Moore ini adalah



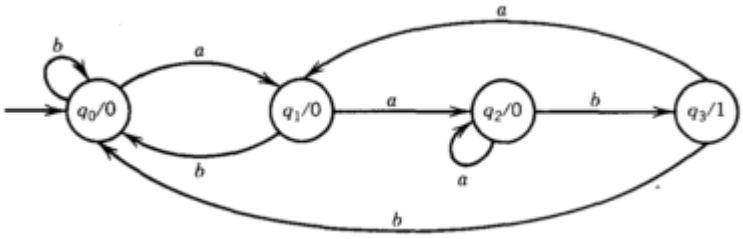
Di mesin Moore, begitu banyak informasi ditulis di dalam State, untuk tanda minus yang menunjukkan state awal. Atau state awal dengan tanda panah ditunjukkan di atas. Mesin FA ini tidak memiliki state akhir, karena string input akan menghasilkan string output dan dapat berakhir dalam keadaan apa pun setelah melakukan tricing pekerjaan. Mari kita telusuri operasi mesin FA ini pada input string abab. mesin ini di start qo, yang secara otomatis mencetak karakter 1. kemudian membaca input huruf pertama dari string input, yang merupakan a dan menuju q1. state ini mencetak 0. Input selanjutnya huruf b, dan looping kembali ke state q1. Berada di q1, sekali lagi, kita mencetak 0 lagi. Kemudian membaca a, pergi menuju ke q3, dan mencetak 1. Selanjutnya baca b, selanjutkan ke q2, dan cetak 0. Ini adalah akhir dari proses. Maka mesin mencetak secara berurutan menghasilkan 10010. Jika mesin more diberikan onput, a, aa, aab, aabb, aabba, maka mesin tersebut menghasilkan sebagimana table input output sebagai berikut,

No.	Input	Output
1	a	10
2	aa	101
3	aab	1010
4	aabb	10101
5	aabba	101011

No.	Input	Output
1	b	11
2	ba	111
3	bab	1110
4	baba	11101
5	babaa	111010

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

Contoh 6.2. Mesin Moore



Input dan output mesin moore dapat dilihat sebagaimana table berikut,

Input String		a	a	a	b	a	b	b	a	a	b	b
State	q_0	q_1	q_2	q_2	q_3	q_1	q_0	q_0	q_1	q_2	q_3	q_0
Output	0	0	0	0	1	0	0	0	0	0	1	0

7.3 Mesin Mealy

Mesin berikutnya adalah mesin Mealy, FA lain yang disebut mesin Mealy. Mesin Mealy seperti mesin Moore kecuali yang membedakan adalah outpunya yaitu berada disisi. Dan kita akan lihat bagaimana operasi mesin mealy ketika mendapatkan input string.

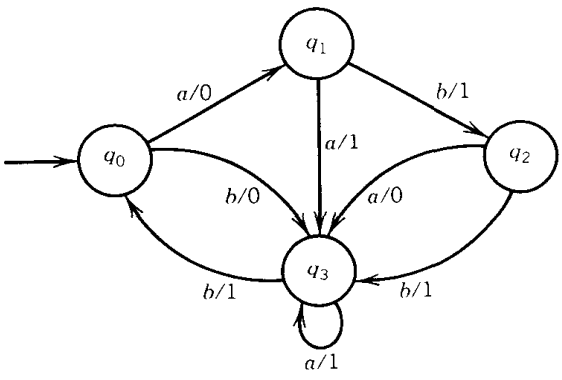
Definisi

a. Mesin Mealy adalah kumpulan dari empat hal:
b. Suatu himpunan State berhingga qo, q1, q2 . . . di mana qo ditunjuk sebagai state awal.
c. Alfabet huruf : Σ {a, b, ... } untuk membentuk string input.
d. Alfabet karakter output Y = {x, y, z . . . }

Representasi state diwakili oleh lingkaran kecil dan sisi terarah menunjukkan transisi antar state. Setiap sisii diberi label dengan simbol majemuk berbentuk i/o di mana i adalah huruf input dan o adalah karakter output. Setiap state harus memiliki tepat satu sisi output untuk setiap huruf input yang mungkin sisi terarah yang ditempuh ditentukan oleh input huruf i; saat disisi harus mencetak karakter output 0.

Contoh Mesin Mealy

Gambar berikut mewakili mesin Mealy:



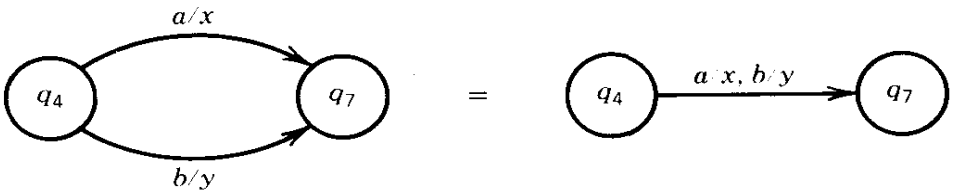
Jika mesin beroperasi datang dari state qo dengan membaca atau mendapatkan input b, maka mesin mencetak 0. Jika kita sampai di sana dari q1, dengan jalan a, kami mencetak 1.. Mari perhatikan dan di telusuri operasinya mesin mealy ini jika dapat input string aaabb. Mesin ini harus mulai dari state,q0. Huruf input pertama adalah a, yang membawa kita ke q1, dan dicetak 0. Huruf kedua adalah a, yang mengambil ke q3 dan mencetak 1. Yang

ketiga huruf adalah a, yang mengulang kita kembali ke q3 dan mencetak 1. Huruf keempat adalah b, yang membawa kita kembali ke qo dan mencetak 1. Huruf kelima adalah b, yang membawa kita ke q3 dan mencetak 0. String output untuk input ini adalah 01110. Input dan output mesin mealy dapat dilihat pada table sebagaimana berikut, pertama mesin membaca input a print 1, kemudian mesin bergerak membaca b print output 1 sekrang berada di state q2, kemudian membaca b yang ke dua print 1, kita berada pada state q3, kemudian input a print 1, masih berada di state q3 siap membaca b,yang terakhir print b, sehingga secara berurutan maka mesin diberikan inputan abbab mmenghasilkann output 01111, pada mesin mealy ini jika mesin diberikan input panjang stringnya N maka outputnyapanjang stringnya juga N, beda dengan mesin moore jika input stringnya N, maka output panjang stringnya adalan N+1

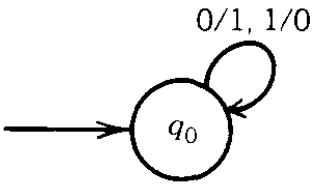
No.	Input	Output
1	a	0
2	Ab	01
3	Abb	011
4	Abba	0111
5	abbab	01111

Perhatikan bahwa dalam mesin Mealy, string output memiliki jumlah yang sama karakter sebagai string input. Seperti mesin Moore, Mesin Mealy tidak mendefinisikan bahasa, sehingga tidak memiliki state akhir.

Jika ada dua sisi yang arahnya sama di antara pasangan yang samavdi state, dapat menggambar satu panah dan mewakili pilihan label dengan koma biasa.



Contoh paling sederhana dari mesin Mealy yang berguna adalah mesin yang mencetak komplemen l dari string bit input. Ini berarti bahwa memproduksi string bit yang memiliki I di mana pun string input memiliki 0 dan 0 di mana pun input memiliki 1. Misalnya, input 101 harus menjadi output 010. Salah satu mesin yang melakukan ini adalah:



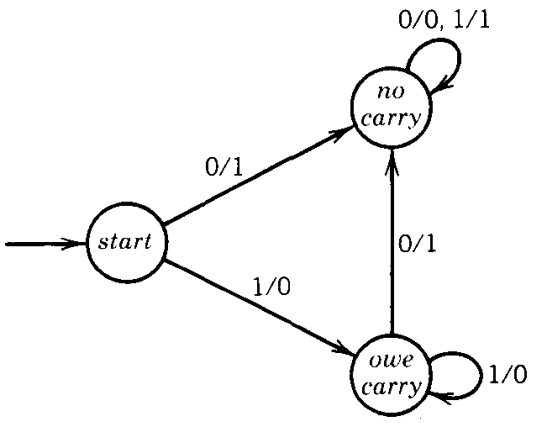
Jika inputnya adalah 001010, outputnya adalah 110101. Ini adalah kasus di mana input alfabet dan alfabet outpunya keduanya {0, 1}.

Contoh

Sekarang mempertimbangkan mesin Mealy yang disebut increment mesin yang mengasumsikan bahwa inputnya adalah bilangan biner dan mencetak bilangan biner. Diasumsikan bahwa string bit input adalah angka biner yang dibelakangnya bit output, yaitu, angka digit pertama (kemudian digit ke 2, digit ke 4 . . .).

Mesin memiliki tiga State: start, owe-carry, no-carry. State owe carry merupakan representasi limpahan ketika dua bit sama dengan 1 ditambahkan dan cetak 0 dan membawa angka biner 1. Dari state awal membaca bit pertama. Jika mesin membaca 0, kemudian mencetak 1. Jika membaca 1, mencetak 0. Jika pada suatu saat dalam proses berada dalam state no-carry, mencetak bit berikutnya dan tetap di state no-carry. Namun, jika pada beberapa proses berada di state owecarry, situasinya berbeda. Jika membaca 0, kemudian mencetak 1 dan menuju state nocarry. Jika berada di state owecarry dan kemudian membaca 1, mencetak 0 dan looping ke state owecarry. Gambaran lengkap untuk mesin ini adalah:

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress



Mari kita lihat bagaimana mesin ini beroperasi pada representasi biner angka decimal sebelas yaitu, 1011. String diinputkan ke dalam mesin biner 1101. Baca biner 1 yang pertama menyebabkan biner 0 dicetak dan menuju ke state owecarry. Lalu kemudian membaca biner 1 berikutnya menyebabkan biner 0 dicetak dan looping kembali ke owecarry. Lalu kemudian input berikutnya membaca biner 0 dan menyebabkan biner 1 dicetak menuju ke state no-carry. tinggal berikutnya, yaitu biner 1, dicetak, saat diinputkan, pada loop no-carry. Sehingga output Stringnya adalah biner 0011, dan, Jika mesin kita inputkan bilangan dua belas binernya 1100. maka melalui proses yang panjang dan akan menghasilkan output biner 0010 Mesin ini memiliki properti mesin Mealy yang khas yaitu string output persis sepanjang string input. Ini berarti bahwa jika menjalankan dengan input 1111 kita akan mendapatkan 0000. Kita harus menafsirkan state owecarry sebagai situasi luapan jika sebuah string pernah berakhir di sana.

Ada hubungan antara mesin Mealy dan rangkaian sekuensial (yang akan didiskusikan pada bab akhir ini) yang menjadikannya sangat berharga di dalam komponen Teori Komputer. Dua contoh yang baru saja disajikan juga berharga untuk komputasi. Setelah kami memiliki incrementer, yaitu dapat membangun sebuah mesin yang dapat melakukan penjumlahan bilangan biner, dan kemudian kita dapat gunakan mesin complementing yang pertama untuk membuat mesin pengurangan berdasarkan prinsip berikut:

Jika a dan b adalah untaian bit, maka pengurangan a - b dapat dilakukan dengan (1) menambahkan komplemen dari b ke a mengabaikan kelebihan digit apa pun dan (2) bertambah hasil dengan 1.

14 - 5 (decimal) = 1110 - 0101 (binary)

= 1110 + l's complement of 0101 + 1 (binary)

= 1110 + 1010 + 1 (binary)

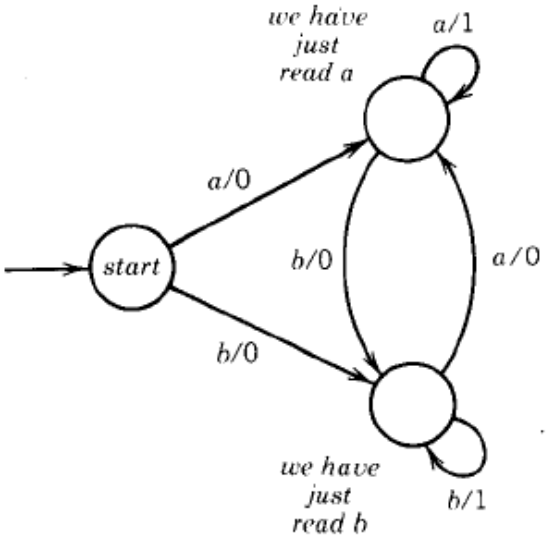
= [1] 1001 binary -- (dropping the [1]) = 9 (decimal)

18 - 7 =10010 - 00111 -- 10010 + 11000 + 1

= [1]01011 -- 01011 = 11 (decimal)

cara yang sama dalam notasi desimal, jika kita menggunakan komplemen 9, yaitu adalah, ganti setiap digit d di angka kedua dengan digit (9 - d). Sebagai contoh, 46 - 17 ---> 46 + 82 + 1 = [1)29 ---> 29.

Meskipun mesin Mealy tidak menerima atau menolak string input, itu dapat mengenali bahasa dengan membuat string outputnya menjawab beberapa pertanyaan tentang input. Kami telah membahas sebelumnya bahasa semua kata yang memiliki huruf double di dalamnya. Mesin Mealy di bawah ini akan mengambil string beberapa a dan b dan cetak string beberapa 0 dan l sedemikian rupa sehingga jika karakter keluaran ke-n adalah 1 artinya huruf yang diinput ke-n adalah yang kedua pada pasangan huruf double. Misalnya ababbaab menjadi 00001010 dengan l di posisi yang kedua dari setiap pasangan huruf berulang



Ini mirip dengan mesin Moore yang mengenali jumlah kemunculan dari substring aab. Mesin ini mengenali terjadinya aa atau bb. Perhatikan bahwa kata tiga huruf aaa menghasilkan output 011 karena huruf kedua dan ketiga adalah bagian belakang dari sepasang huruf a ganda.

Definisi

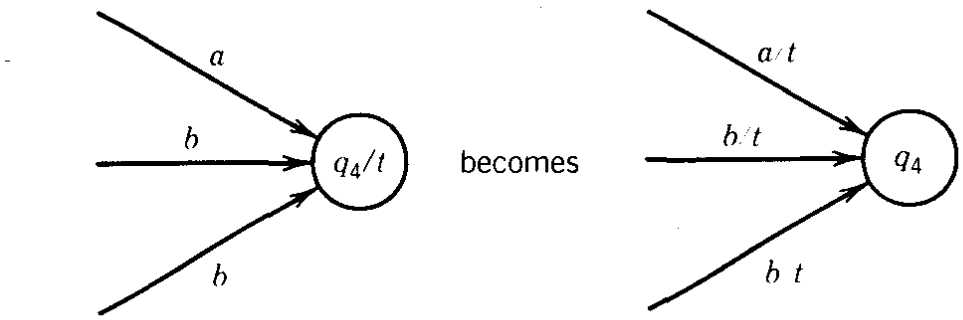
Mesin Mealy Me dan mesin Moore Mo,dikatakan bahwa kedua mesin ini adalah setara jika untuk setiap string input dan string output dari Mo persis x digabungkan dengan output dari Me. dibuktikan bahwa untuk setiap mesin Moore ada mesin Mealy yang setara dan untuk setiap Mesin Mealy ada yang setara dengan mesin Moore. kemudian dapat mengatakan bahwa kedua jenis mesin tersebut benar-benar setara.

THEOREMA

Jika Mo adalah mesin Moore, maka ada mesin Mealy Me yang setara untuk itu.

BUKTI

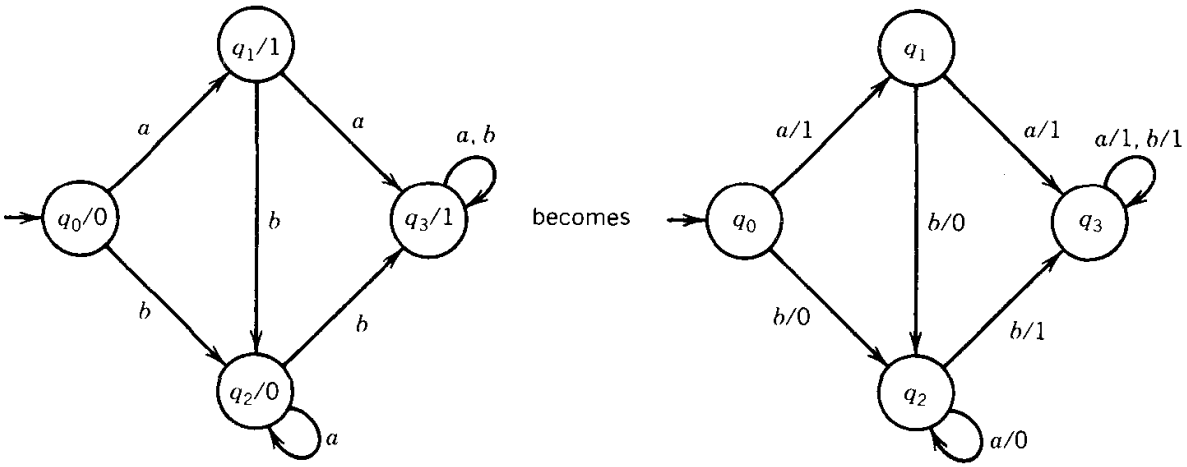
Buktinya dengan algoritma. State q4. Ini memberikan instruksi untuk mencetak karakter t. semua sisi yang yang menuju state ini. Masing-masing diberi label input. Mari kita ubah ini. kita beri label ulang semua sisi yang masuk ke state q4. Jika sebelumnya diberi label input a atau b atau c . . . , sekarang sisi diberi label a/t, b/t, b/t. . . dan kemudian pada dalam state q4 output label t di hapus ni berarti bahwa kita akan mencetak t pada sisi yang masuk sebelum mereka menuju state q4., sebagaimana gambar



Contoh di bawah ini,

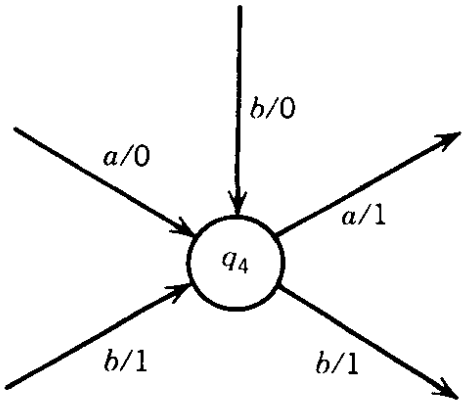
Mesin Moore diubah menjadi mesin Mealy

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

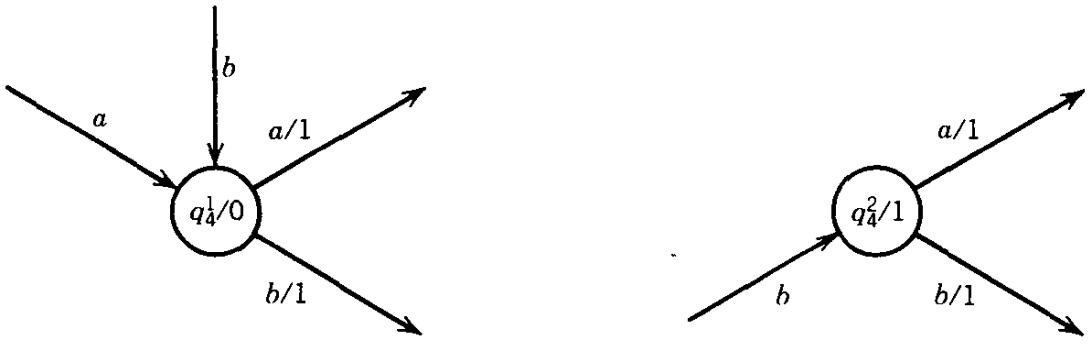


Untuk setiap mesin Mealy Me ada mesin Moore Mo yang setara

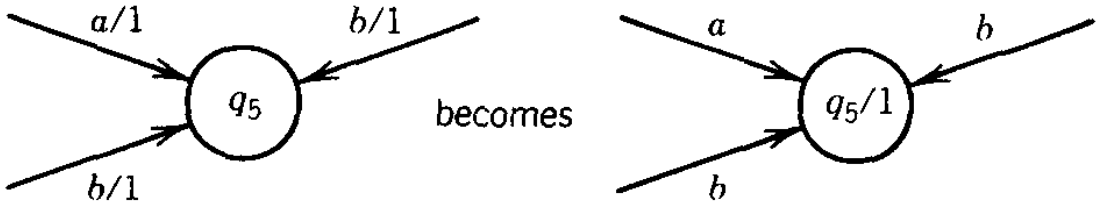
Dua I/O sisi berbeda menuju state yang sama, seperti dalam contoh ini.



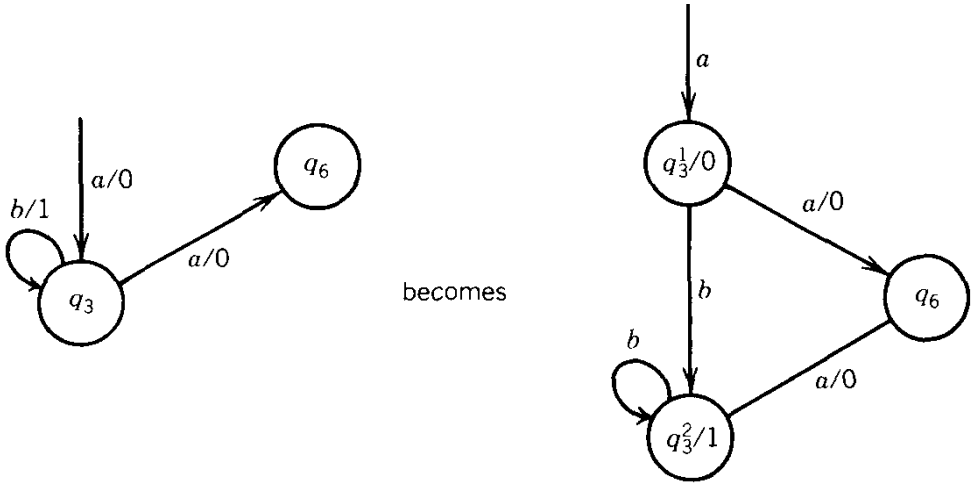
Pada sisi transisi a/0 menuju ke q41 dan pada label sisi transisi b/1 transisi menuju state q42. Label transisi pada sisi b/0 juga menuju state q41. Didalam state ini termasuk instruksi cetak/output q41/0 dan q42/1. Sementara sis trasisi yang keluar dari state q4, masii tetap yaitu a/1 dan b/1, sebagaimana gambar dibawah ini :



Untuk Instruksi cetak 0 atau 1 di dalam state, Jika semua sisi masuk ke dalam state memiliki instruksi cetak /output yang sama, maka cukup memindahkannya instruksi cetek ke state.



Salah satu yang menarik dari algoritma ini adalah bahwa sisi transisi yang looping di mesin mealy dapat menjadi satu sisi yang bukan looping dan yang looping di mesin Moore. Misalnya



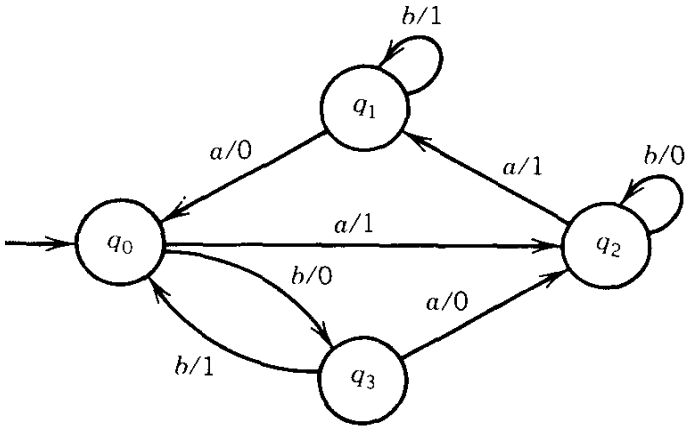
Pada contoh di atas adalah bahwa label transisi a/0 menuju state q3 yang mencetak 0. di sebut state q31/0. Looping transisi berlabel b/1 di state q3 harus menuju state q3 yang mencetak 1. Di sebut state q32/1. Ketika pada sisi transisi a/0 menuju state q3, kita masukkan kedalam q31/0, tapi harus kembali menuju state q32/1. Dengan input transisi b, demikian pula untuk input/output b/1 menuju state q3, dimasukkan menjadi tramsisi input b menuju state q32/1.

Semestara dari state q3 menuju state q6 dengan I/0, a/0 menjadi q31/0. Menuju state q6 dengan input/output a/0 dan q32/1 transisi menuju state q6 dengan input/output a/0 juga. Sehingga sebagaimana disebutkan diatas bahwa mesin mealy equevalen dengan mesin Moore,

Me =Mo

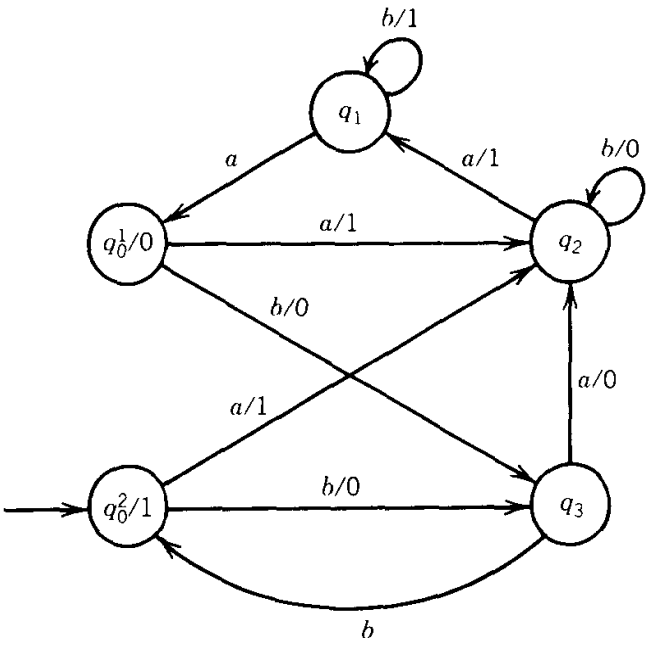
Sebagai contoh

Mesin mealy sebagai berikut,

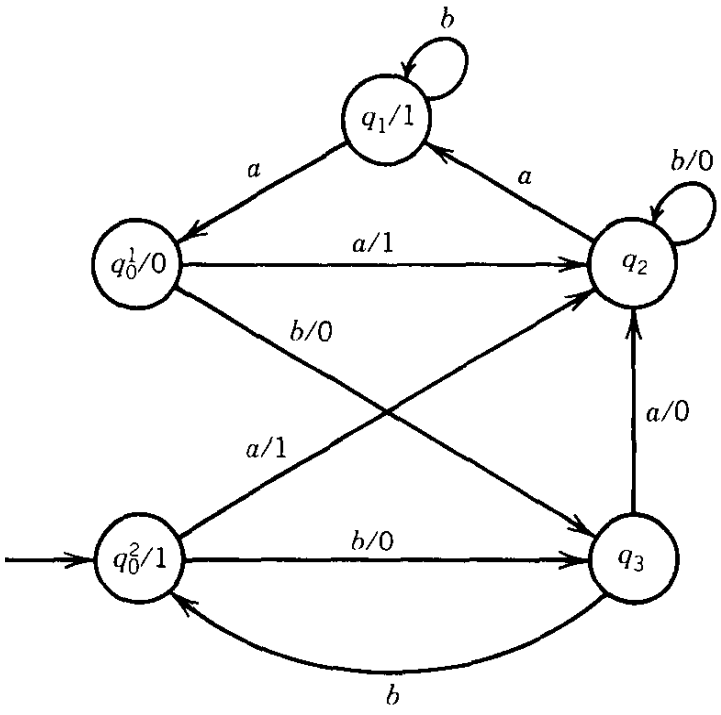


Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

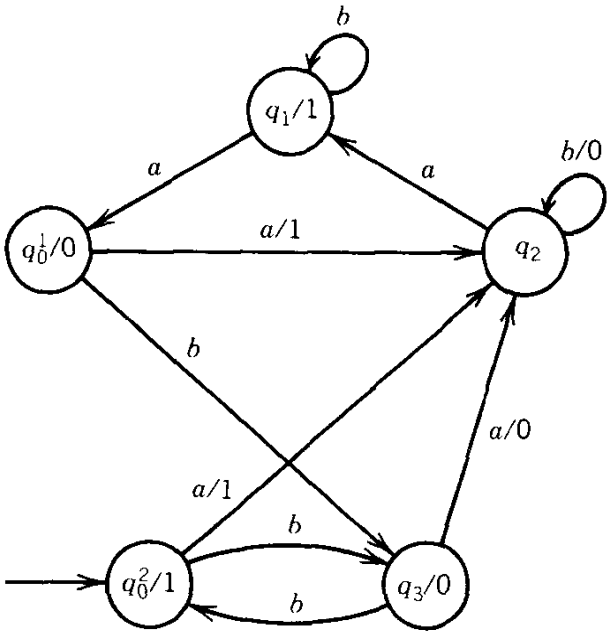
Proses pertama



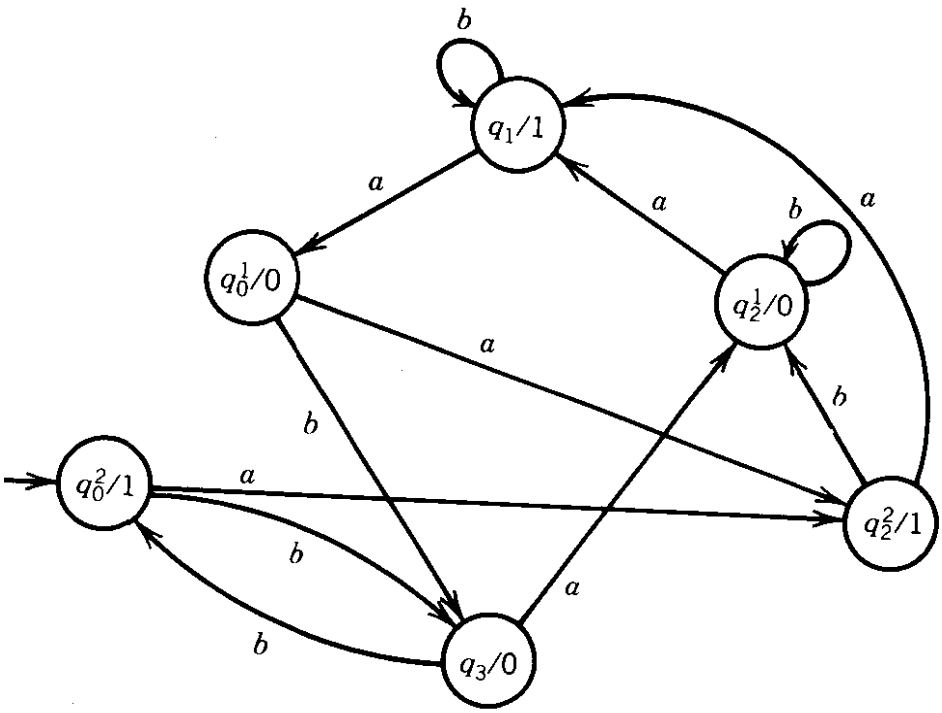
Proses ke dua menjadi,



Ketiga,



Maka menghasilkan mesin Moore sebagai berikut,

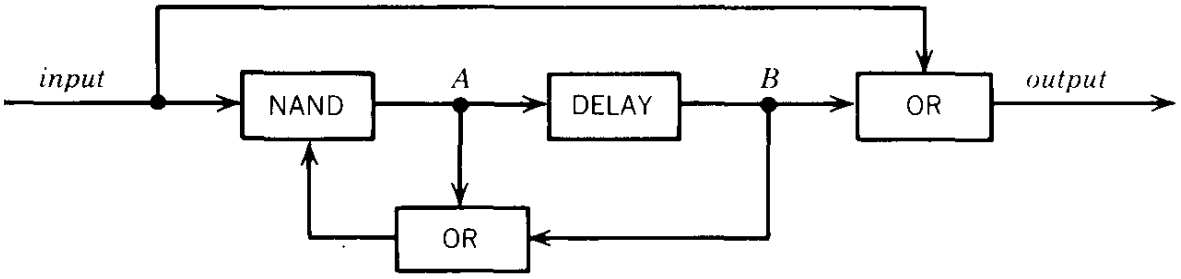


Contoh rangakaian sekuensial

Perhatikan contoh rangkaian sekuensial sederhana.

Rangkaian NAND berarti outpunya membawa rangkaian carrier output Boolean rangkaian AND inputnya. Output dari kotak DELAY adalah sama dengan inputan sebelumnya. Penundaan transmisi sinyal di sepanjang kabel dengan satu langkah (pulsa clock). DELAY kadang-kadang disebut D flip-flop. AND dan OR seperti biasa. Arus mengalir dilambangkan dengan nilai 1, tidak ada arus yang mengalir disebut nilainya sebesar 0.

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress



Pada gambar ini mengidentifikasi empat State adanya atau tidakadanya arus yang mengalir padarangkaian titik A dan B.

qo adalah A = 0 B = 0
q1 adalah A = 0 B = 1
q2 adalah A = 1 B = 0
q3 adalah A = 1 B = 1

Sedemikian rupa Operasi rangkaian ini sehingga setelah input 0 atau 1 keadaan berubah sesuai dengan aturan berikut:

yang baru B = yang lama A
Baru A = (input) NAND (lama A OR lama B
Output = (input) OR (old B)

Pada berbagai pulsa diskrit dari input clock waktu diterima, state berubah, dan output dihasilkan

Misalkan berada di state q0 dan menerima input 0:

baru B = lama A = 0

baru A = (input) NAND (lama A OR lama B)

= (0) NAND (0 OR 0)

= 0 NAND 0

= 1

output = 0 OR 0 = 0

new state adalah q2 (dimana yang baru A = 1, baru B = 0).

Jika berada di state qo dan menerima input 1

baru B = lama A = 0

baru A = 1 NAND (0 OR 0) = 1

output = 1 OR 0 = 1

new state pada State q2 (dimana yang baru A = 1 and baru B = 0).

jika di state q1, dan menerima input 0:

baru B = lama A = 0

baru A = 0 NAND (0 OR 1) = I

output = 0 OR 1 = I

Pada State baru q2

Jika berada pada state q2 dan menerima input 0

Baru B = lama A = 1

Baru A = 0 NAND (1 OR0) = 1

output = 0OR0 = 0

Pada State baru q3 (dimana baru A = 1, baru B = 1)

Jika berada dalam q2 dan menerima input 1 ;

Baru B = lama A = 1

baru A = 1 NAND (1 OR 0) = 0

output = I OR0 = I

Pada new state is q1 .

Jika state berada pada state q3 dan menerima input 0 :

Baru B = lama A = 1

baru A = 0 NAND (1 OR 1) = 1

output = 0 OR 1 = 1

Pada new state is q3.

Jika berada di state q3 dan menerima input 1:

baru B = lama A = I

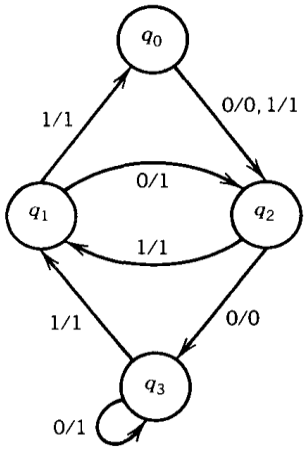
baru A = 1 NAND (1 OR 1) = 0

output = I OR 1 = I

Pada new state is q1 .

	After input 0		After input 1	
Old state	**New state**	**Output**	**New state**	**Output**
q_0	q_2	0	q_2	1
q_1	q_2	1	q_0	1
q_2	q_3	0	q_1	1
q_3	q_3	1	q_1	1

Operasional arus sequential adalah equivalent dengan mesin Mealy sebagaimada graph berikut,

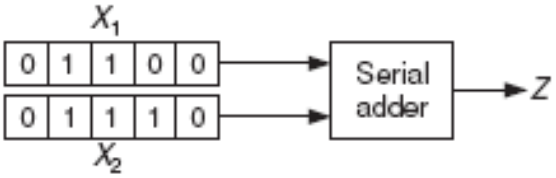


input string input 011011 akan menyebabkan urutan outputnya adalah 111011.

Contoh Aplikasi rangkaian sekuensialyang lain

Kita perhatikan bahwa sirkuit switching kombinasional di mana output nilai adalah fungsi dari nilai masukan arus rangkaian. Di sebagian besar system digital, sirkuit tambahan diperlukan yang mampu menyimpan data dan informasi juga beberapa data ini dapat melakukan operasi logika matematika. Nilai output dari sirkuit pada waktu tertentu adalah fungsi dari nilai input yang disimpan. Sirkuit seperti itu disebut sirkuit sekuensial dri Mesin finite automata yaitu model mesin abstrak yang menggambarkan jaringan mesin sekuensial sinkron. Merupakan dasar untuk untuk memahami dan pengembangan berbagai struktur komputasi. Keterbatasan, dan struktur mesin finite automata with output

Sering juga kita menjumpai penggunaan berbagai rangkaian sekuensial. Kontrol lift yang mengangkat orang masuk ke lift; sistem lampu lalu lintas di jalan raya, kereta api, dan kereta bawah tanah,kunci brankas yang mengingat nomor kombinasi berdasarkan kunci nomor urutan, semua ini adalah contoh operasional system rangkaian sekuensial.



Block diagram dari serial binary adder.

Table state

Ditunjukkan pada gambar Block diagram dari serial binary adder, adalah sirkuit sinkron dengan dua input, X1 dan X2, membawa dua nomor biner yang akan ditambahkan dan satu output, Z, yang mewakili jumlah. Panjang tetap pada urutan 0 dan 1 diumpankan ke input dan diperoleh pada output. Penambahan itu harus dilakukan secara berurutan: digit angka paling signifikan X1 dan X2 sampai dengan di terminal input yang sesuai pada waktu t1; satu satuan waktu kemudian,digit signifikan berikutnya hingga terkecil sampai pada di terminal input; dan seterusnya. Interval waktu itu antara kedatangan dua digit input berurutan ditentukan dengan frekuensi jam sirkuit. Mesin akan berasumsi bahwa penundaan dalam sirkuit kombinasional kecil sehubungan dengan periode jam (yang merupakan kebalikan dari frekuensi clock) dan, sebagai akibatnya, jumlah digit tiba di terminal Z segera setelah kedatangan digit input yang sesuai di terminal masukan. Mesin akan menunjukkan dengan X dan Z urutan input dan output, masing-masing, dan dengan x dan z simbol input dan output pada titik waktu tertentu. Mungkin keadaan ini sering kecenderungan menekankan waktu yang tepat di mana nilai input atau outputnya. Dalam kasus seperti itu, notasi x(ti), z(ti) akan digunakan.

$$\begin{array}{rccccc} & t_5 & t_4 & t_3 & t_2 & t_1 \\ & 0 & 1 & 1 & 0 & 0 = X_1 \\ + & 0 & 1 & 1 & 1 & 0 = X_2 \\ \hline & 1 & 1 & 0 & 1 & 0 = Z \end{array}$$

Pemeriksaan korelasi antara nilai input dan yang dibutuhkan nilai output mengungkapkan perbedaan mendasar antara rangkaian kombinasional dan penambah biner serial. Sedangkan pada rangkaian kombinasi nilai keluaran pada waktu ti didefinisikan secara unik oleh nilai input di ti , dalam penambah serial berbeda

nilai output diperlukan untuk kondisi input yang identik. Misalnya, pada t1 dan t5 nilai input adalah x1x2 = 00, tetapi nilai output yang diperlukan adalah z = 0 dan z = 1, masing-masing. Demikian pula, pada t3 dan t4 nilai inputnya adalah x1x2 = 11 sedangkan nilai output yang diinginkan adalah 0 dan 1, masing-masing. Oleh karena itu, jelas bahwa nilai keluaran penambah serial tidak dapat ditentukan hanya dalam hal nilai input eksternal, dan prosedur desain yang berbeda harus digunakan.

Mengikuti aturan aritmatika biner dasar, terbukti bahwa nilai keluaran pada waktu ti adalah fungsi dari nilai masukan x1 dan x2 pada waktu itu dan carry yang dihasilkan pada ti−1.

	NS, z			
PS	$x_1x_2 = 00$	01	11	10
A	$A, 0$	$A, 1$	$B, 0$	$A, 1$
B	$A, 1$	$B, 0$	$B, 1$	$B, 0$

Gambar 7.1: State table for a serial binary adder

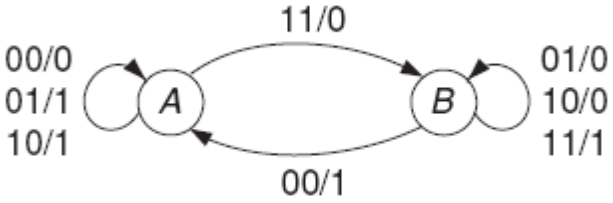
Carry ini (yang mungkin memiliki salah satu dari dua nilai 0 atau 1) pada gilirannya tergantung pada nilai input pada ti−1 dan pada carry yang dihasilkan pada ti−2, dan seterusnya. Oleh karena itu penambah harus dapat melestarikan informasi mengenai nilai inputnya sejak diatur ke dalam operasi ke waktu ti. Namun, karena waktu mulai mungkin sudah lama berlalu, tidak mungkin untuk melestarikan seluruh sejarah nilai input. Oleh karena itu kami mencari hubungan yang berbeda antara nilai input x1(ti) dan x2(ti) dan nilai output z(ti), sebagai berikut.

Dalam kasus penambah serial, kita dapat membedakan dua kelas input sebelumnya sejarah, yang satu menghasilkan produksi carry 0 dan yang lainnya menghasilkan a carry 1. Kelas-kelas ini akan disebut state internal (atau hanya state) dari penambah. Dengan "menghafal" nilai carry, penambah benar-benar menunjukkan beberapa "jejak" dari nilai input masa lalunya, setidaknya sejauh pengaruhnya terhadap respon terhadap nilai input saat ini.

Biarkan A menyatakan State penambah di ti jika carry 0 dihasilkan di ti −1, dan misalkan B menyatakan status penambah di ti jika carry 1 dibangkitkan pada ti −1.Kami mengacu pada keadaan penambah pada saat nilai input saat ini adalah diterapkan padanya sebagai keadaan sekarang (PS) dan keadaan tempat penambah pergi, sebagai hasil dari nilai carry yang baru (belum tentu berbeda), sebagai next state (NS). Nilai keluaran z(ti) adalah fungsi dari nilai masukan x1 (ti) dan x2(ti) dan keadaan penambah pada waktu ti . Status penambah berikutnya hanya bergantung pada nilai input saat ini dan pada keadaan sekarang. Cara yang nyaman untuk menggambarkan perilaku penambah serial adalah melalui tabel status, seperti yang ditunjukkan pada gambar State table for a serial binary adder

Setiap baris tabel state sesuai dengan state penambah, dan masing-masing kolom ke kombinasi tertentu dari nilai input eksternal x1 dan x2. Setiap entri tabel menunjukkan state di mana transisi dibuat dan nilai outputnya yang terkait dengan transisi ini. Misalnya, jika penambah dalam state A, yaitu, carry saat ini adalah 0, dan menerima kombinasi input x1x2 = 11 maka akan ke state B, yang sesuai dengan carry 1, dan menghasilkan nilai output z = 0. Entri tabel yang tersisa dapat diverifikasi secara langsung dan, karena tabel berisi delapan entri, sesuai dengan delapan state kombinasi dan nilai input, itu merupakan sepenuhnya menentukan serial adder.

Seringkali lebih mudah untuk menggunakan graph berarah sebagai lawan dari table state. Graph seperti itu, yang ditunjukkan pada Gambar 9.2, dikenal sebagai diagram state (atau state graph). Simpul dan busur berarah dari graph sesuai dengan state dari adder dan transisi statenya masing-masing



Gambar 7.2: State diagram untuk serial adder.

Label dari busur berarah menentukan nilai input dan nilai output yang sesuai; misalnya, 10/0 menyatakan kondisi x1 = 1, x2 = 0, dan z = 0. Jelas, kedua keadaan diagram dan tabel keadaan memberikan informasi yang sama mengenai operasi adder (penambah), dan satu dapat diperoleh langsung dari yang lain. Sementara di banyak kasus

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

representasi ini sama-sama cocok, dalam beberapa aplikasi automata satu ini mungkin menjadi lebih baik daripada yang lain.

7.4 Latihan Soal

Masing-masing dari berikut ini adalah mesin Moore dengan input alfabet = {a,b} dan output alfabet = {0,1}. Dalam Soal 1 sampai 5, gambarkan mesin-mesin dengan tabel transisi dan output. Dalam Soal 6 sampai 10, buat transisinya dan tabel output yang diberikan pada gambar dibawah ini.

1.

	a	b	Output
q_0	q_1	q_2	1
q_1	q_1	q_1	0
q_2	q_1	q_0	1

2.

	a	b	Output
q_0	q_0	q_2	0
q_1	q_1	q_0	1
q_2	q_2	q_1	1

3.

	a	b	Output
q_0	q_0	q_1	1
q_1	q_0	q_2	0
q_2	q_2	q_2	1
q_3	q_1	q_1	0

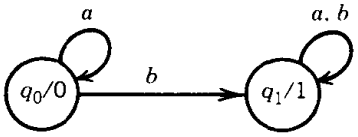
4.

	a	b	Output
q_0	q_3	q_2	0
q_1	q_1	q_0	0
q_2	q_2	q_3	1
q_3	q_0	q_1	0

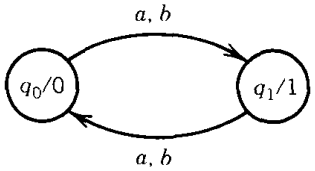
5.

	a	b	Output
q_0	q_1	q_2	0
q_1	q_2	q_3	0
q_2	q_3	q_4	1
q_3	q_4	q_4	0
q_4	q_0	q_0	0

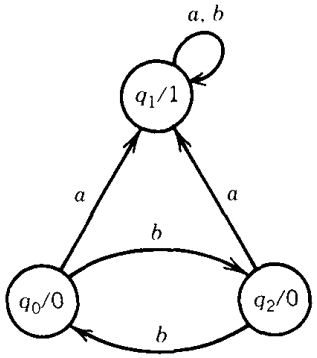
6.



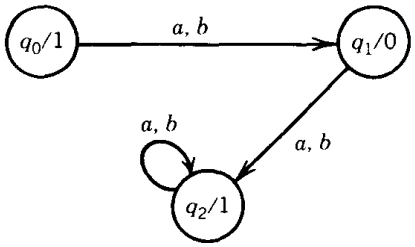
7.



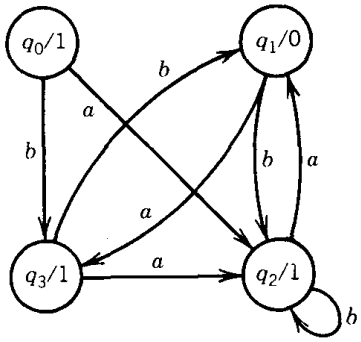
8.



9.



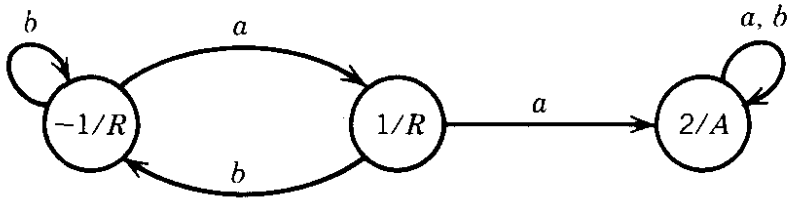
10.



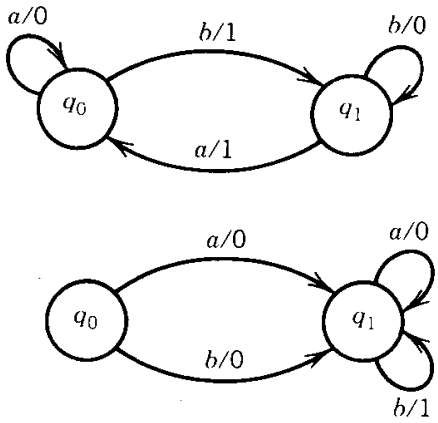
11. Pada masing-masing mesin Moore di Soal 1 sampai 10, operasikan mesin jika diberikan input urutan aabab. Apa output masing-masing?
12. Meskipun mesin Moore tidak mendefinisikan bahasa string yang diterimanya sebagai input, kita masih dapat menggunakannya untuk mengenali bahasa di pengertian berikut. Kita dapat mengatur bahwa karakter terakhir yang dicetak oleh mesin adalah A untuk menerima atau R untuk menolak. Misalnya, mesin Moore berikut akan mengenali bahasa semua kata dari regular ekpresi

    $$(a + b)^*aa(a + b)^*$$

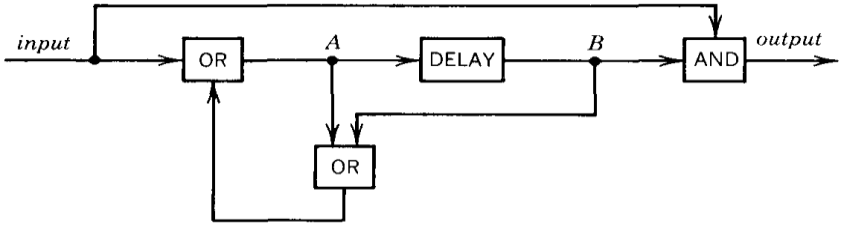
    artinya bahwa kata-kata ini dan hanya kata-kata ini akan menyebabkan karakter terakhir yang dicetak oleh mesin menjadi A



    Tunjukkan bahwa semua bahasa yang ditentukan oleh ekspresi reguler dapat dikenali oleh mesin Moore
13. Buatlah tabel transisi untuk masing-masing dari empat mesin Mealy yang ditunjukkan



14. Gambarlah mesin Mealy yang ekuivalen dengan rangkaian sekuensial berikut.



Bab 8

Bahasa Reguler

8.1 Theorem

Bahasa yang dapat didefinisikan dengan ekspresi reguler disebut bahasa regular. Dalam bab berikutnya ini membahas pertanyaan penting, "Apakah semua bahasa itu reguler?" Jawabannya tidak. Tapi sebelum mulai membuktikan ini, membahas beberapa sifat dari kelas bahasa. Bahasa reguler dirangkum dalam teorema berikut

Jika bahasa L1 dan L2 adalah bahasa-bahasa regular, maka L1 + L 2 , L1L2 dan LI* juga bahasa reguler L1 + L 2 berarti bahasa yang semua kata baik dalam L1 atau L2. L1L 2, bahasa semua kata yang dibentuk dengan menggabungkan kata dari L dengan kata dari dalam L 2. L1* berarti string yang merupakan rangkaian dari banyak faktor dari L 1. Hasil yang dinyatakan dalam teorema ini sering dinyatakan dengan mengatakan: Himpunan bahasa reguler yang dikatakan union, concatenation, dan Kleene operator star.

Bukti 1 (berdasarkan Ekspresi Reguler)

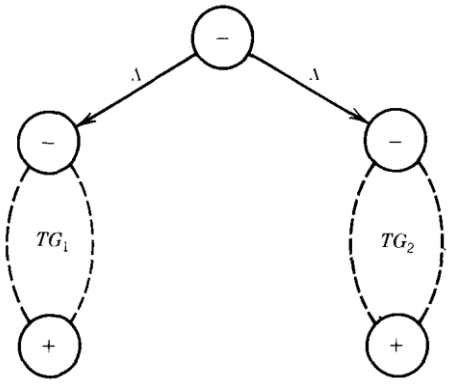
Jika L, dan L2 adalah bahasa reguler, ada ekspresi reguler r1 dan r2 yang mendefinisikan bahasa-bahasa ini. Maka (r1 + r 2) adalah ekspresi reguler yang mendefinisikan bahasa L1 + L2. Bahasa L1L 2 dapat didefinisikan dengan ekspresi reguler r1r 2. Bahasa L1* dapat didefinisikan dengan ekspresi reguler (r1)*. Oleh karena itu, ketiga rangkaian kata ini dapat didefinisikan dengan ekspresi reguler dan begitu juga bahasa reguler mereka sendiri.

Pembuktian Teorema di atas menggunakan fakta bahwa L1 dan L2 pasti didefinisikan oleh ekspresi reguler jika itu adalah bahasa reguler. Bahasa reguler juga dapat didefinisikan dalam mesin, dan seperti yang terjadi pada mesin juga dapat digunakan untuk membuktikan teorema ini.

Pembuktian dengan Mesin

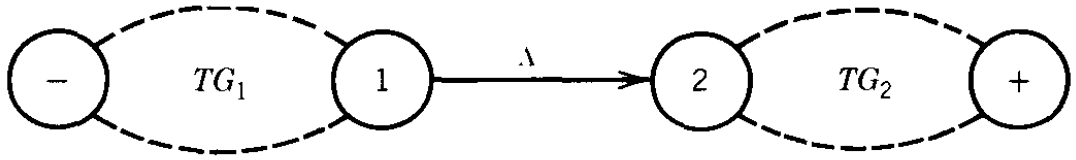
Karena L, dan L2 adalah bahasa reguler, pasti ada TG yang menerimanya. Biarkan TG1 menerima L, dan TG2 menerima L2 . Mari kita asumsikan lebih lanjut bahwa TG1 dan TG 2 masing-masing memiliki state awal yang unik dan state akhir terpisah yang unik. Jika ini tidak demikian aslinya, dapat memodifikasi TG sehingga menjadi sebagai berikut,

TG ini menerima bahasa L1 + L2:



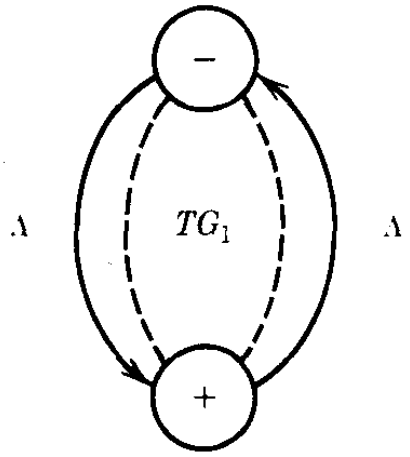
Gambar 8.1: TG ini menerima bahasa L1 + L2

TG menerima bahasa L1L22:



Gambar 8.2: TG menerima bahasa L1L22

TG menerima bahasa L1*:



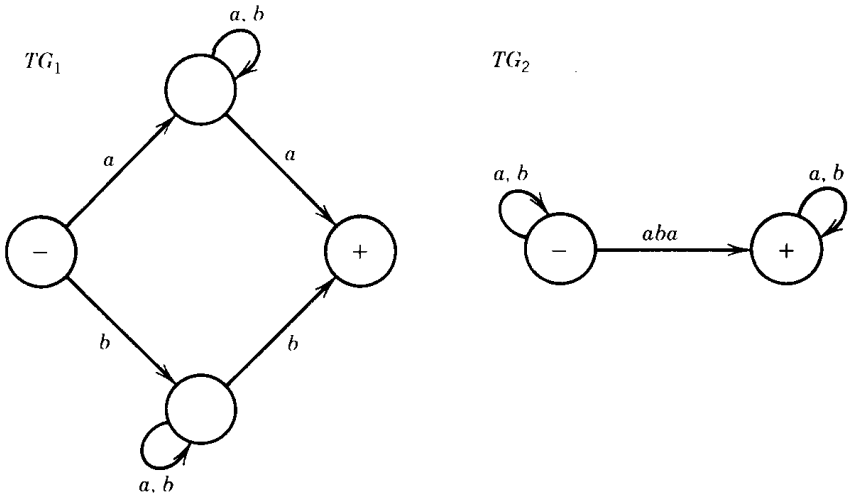
Gambar 8.3: TG menerima bahasa L1*

CONTOH

Misalkan alfabetnya = {a,b}.

L1 = semua kata dari dua huruf atau lebih yang dimulai dan diakhiri dengan huruf yang sama dan

L2 = semua kata yang mengandung substring aba Untuk bahasa ini , menggunakan TG berikut dan ekspresi regulernya :



r1 r2

a(a + b)*a + b(a + b)*b (a + b)* aba (a + b)*

Bahasa L1 + L2 adalah regular karena dapat didefinisikan dengan regular ekspresi:

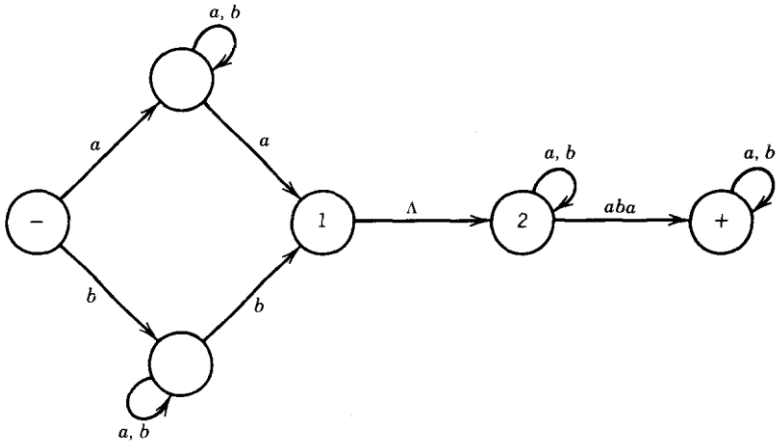
[a(a+b)*a + b(a+b)*b] + [(a+b)* aba (a+b)*]

Maka dapat dibuktikan dengan mesin sebagai berikut

Skarang Bahasa L1L2 adalah reguler karena dapat didefinisikan oleh ekspresi reguler dan mesin TG sebagaimana berikut:

Ekpresinya [a(a+b)*a + b(a+b)*b] [(a+b)* aba (a+b)*

Mesin TGnya yang equevalensi;



Bahasa L1* adalah bahasa reguler karena dapat didefinisikan oleh ekspresi reguler:

[a(a+b)*a + b(a+b)*b]*

TG dari ekpresi regularnya adalah,

Definisi

Jika L adalah bahasa di atas alfabet ∑ kami mendefinisikan L', untuk menjadi bahasa semua untaian huruf dari ∑ yang bukan kata-kata di dalam L.

Banyak penulis menggunakan notasi batang L komplemen untuk menunjukkan pelengkap dari bahasa L, tapi, seperti kebanyakan menulis untuk komputer, kami akan menggunakan lebih banyak bentuk.

Contoh

Jika L adalah bahasa di atas alfabet ∑ = {a ,b} dari semua kata yang memiliki a ganda di dalamnya, maka L' adalah bahasa dari semua kata yang tidak memiliki a ganda.

Penting untuk menentukan alfabet I atau pelengkap L mungkin mengandung "kucing", "anjing", "katak". . , karena ini jelas bukan string di L.

Perhatikan bahwa pelengkap bahasa L' adalah bahasa L. Kita bisa tulis ini sebagai

L" = L

Ini adalah teorema dari Teori Himpunan yang tidak terbatas hanya pada bahasa.

Teorema 11

Jika L adalah bahasa reguler, maka L' juga merupakan bahasa reguler. Dengan kata lain, set bahasa reguler closed under complementation.

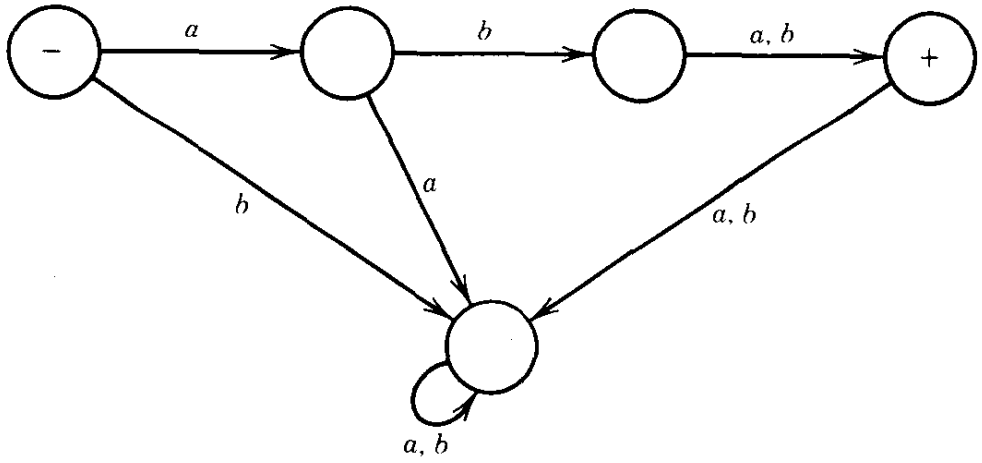
Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

Bukti

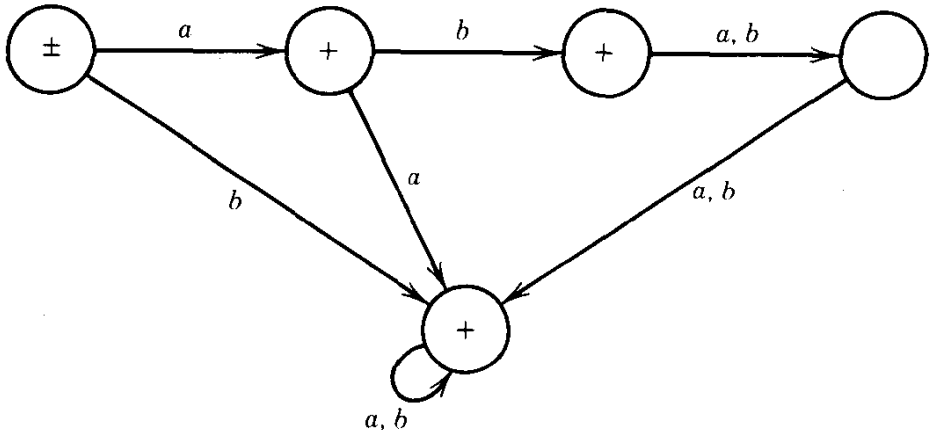
Jika L adalah bahasa reguler, kita tahu dari Teorema Kleene bahwa ada beberapa FA yang menerima bahasa L. Beberapa state FA ini bersifat final state bagian dan, kemungkinan besar, beberapa tidak. Mari kita membalikkan state akhir masing-masing menyatakan, yaitu, jika itu adalah keadaan final, buatlah itu menjadi state nonfinal, dan jika itu adalah state nonfinal, jadikan itu state final. Jika string input sebelumnya berakhir dengan a state nonfinal, sekarang berakhir pada state final dan sebaliknya. Mesin baru ini kami telah membangun menerima semua string input yang tidak diterima oleh aslinya FA (semua kata dalam L') dan menolak semua string input yang digunakan FA untuk menerima (kata-kata dalam L). Oleh karena itu, mesin ini menerima bahasa yang tepat L'. Jadi menurut Teorema Kleene, L'adalah regular.

Contoh

FA yang hanya menerima string aba dan abb ditunjukkan di bawah ini:



FA yang menerima semua string selain aba dan abb adalah



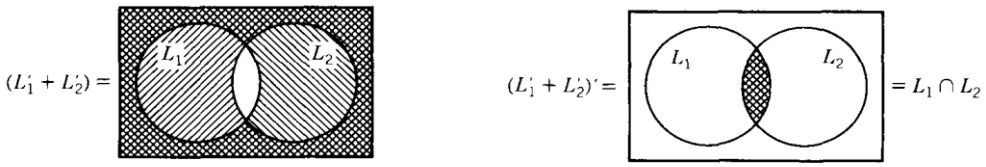
Jika L, dan L2 adalah bahasa reguler, maka L1 ∩ L2 juga merupakan bahasa reguler. Dengan kata lain, himpunan bahasa reguler merupakan intersectin.

Bukti

Menurut Hukum DeMorgan untuk set jenis apa pun (bahasa regular atau bukan):

$$L_1 \cap L_2 = (L_1' + L_2')'$$

Hal ini diilustrasikan oleh diagram Venn di bawah ini.



Artinya bahasa L1 ⋂ L2 terdiri dari semua kata yang tidak baik L,' atau L2'. Karena L1 dan L2 beraturan, maka L1' dan L2' juga demikian. Sejak Ll' dan L2' regular, begitu juga L1' + L2'. Dan karena LI' + L2' regular, maka begitu juga (LI' + L2')', yang berarti L1 ⋂ L2 regular.

Ini adalah kasus "buktinya lebih cepat daripada mata." Ketika kita mulai dengan dua bahasa L, dan L2 , yang dikenal teratur karena mereka didefinisikan oleh FA, menemukan FA untuk L1 ⋂ L2 tidak semudah yang dibuktikan. Jika L, dan L2 didefinisikan oleh ekspresi reguler yang menemukan L1 ⋂ L2 dapat menjadi lebih baik. Namun, semua algoritma yang kita butuhkan untuk konstruksi ini telah dikembangkan.

Contoh

Mari kita kerjakan satu contoh secara lengkap. Kami mulai dengan dua bahasa atas ∑ = {a,b}.

L, = semua string dengan double a

L2 = semua string dengan jumlah a genap.

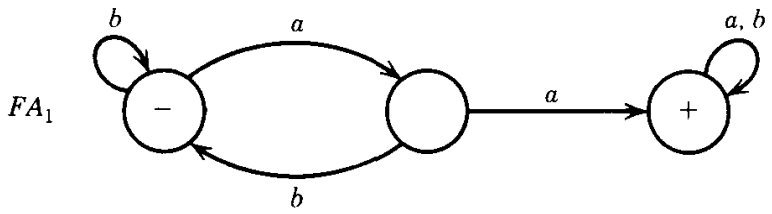
Bahasa-bahasa ini tidak sama, karena aaa ada di L1 tetapi tidak di L2 dan aba ada di L 2 tapi tidak di L1 .

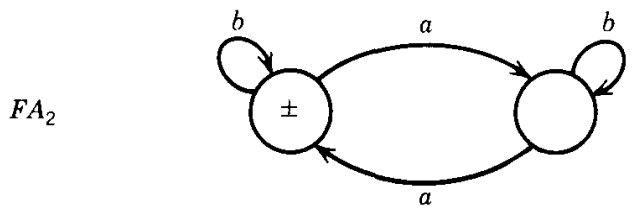
Keduanya adalah bahasa reguler karena didefinisikan sebagai berikut:sebagai ekspresi reguler (antara lain).

r, = (a + b)*aa(a + b)*

r2 = b*(ab*ab*)*

Ekspresi reguler r 2 agak baru bagi kita. Sebuah kata dalam bahasa L2 dapat memiliki beberapa b di depan, tetapi kemudian setiap kali ada a itu seimbang (setelah beberapa b) dengan yang lain a. Ini memberi kita faktor bentuk (ab*ab*). Kata dapat memiliki banyak faktor dalam bentuk ini sesuai keinginannya. Dia dapat diakhiri dengan a atau b. Karena kedua bahasa ini teratur, Teorema Kleene mengatakan bahwa mereka dapat juga ditentukan oleh FA. Dua yang terkecil adalah:





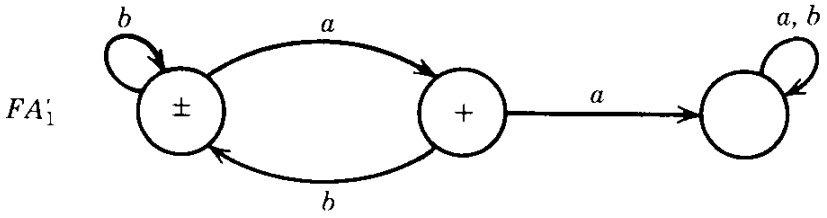
Di mesin pertama kita tetap dalam state awal sampai kita membaca a, kemudian kita pindah ke state tengah. Ini adalah kesempatan kita untuk menemukan ganda sebuah. Jika kita membaca a lain dari string input saat dalam state tengah, kita pindah ke state akhir di mana kita tinggal. Jika kita melewatkan kesempatan kita dan membaca

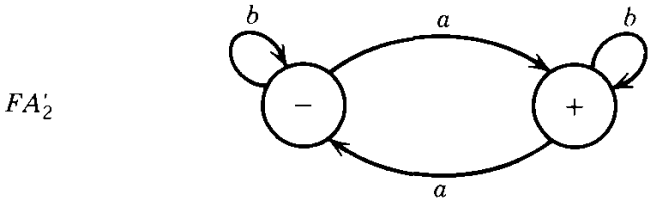
b, kita kembali ke -. Jika kita tidak pernah melewati state tengah, kata memiliki tidak ada ganda a dan ditolak. Mesin kedua beralih dari state kiri ke state kanan atau dari kanan state ke state kiri setiap kali membaca a. Itu mengabaikan semua b. Jika string dimulai di kiri dan berakhir di kiri, itu pasti membuat bilangan genap dari sakelar kiri/kanan. Oleh karena itu, senar yang diterima mesin ini persis yang ada di L2. Sekarang langkah pertama dalam membangun mesin (dan ekspresi reguler) untuk L1 ⋂ L2 adalah untuk menemukan mesin yang menerima complemen bahasa L1' dan L2'. Meskipun tidak diperlukan untuk keberhasilan eksekusi algoritma, deskripsi bahasa dari bahasa-bahasa ini adalah:

L,' = semua string yang tidak mengandung substring aa

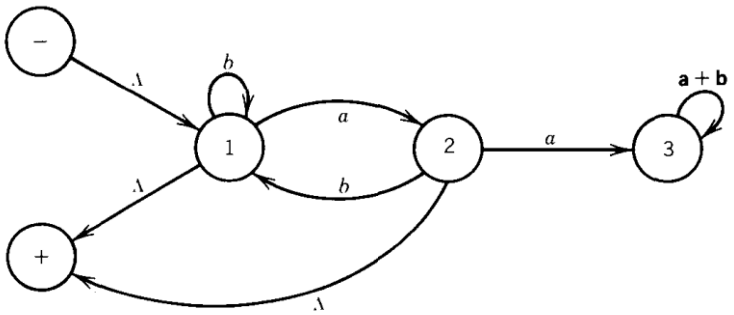
L2' = semua senar yang memiliki bilangan ganjil a's

Dalam pembuktian teorema bahwa pelengkap bahasa biasa adalah biasa kami memberikan algoritme untuk membangun mesin yang menerima bahasa ini. Yang harus kita lakukan hanyalah membalikkan stateakhir dan apa bukanlah keadaan akhir. Mesin untuk bahasa ini kemudian

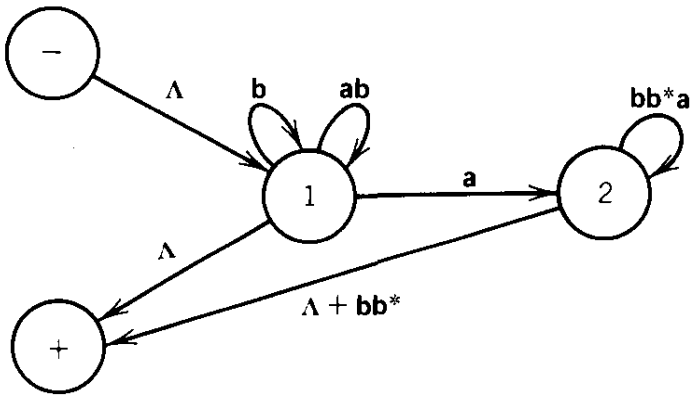




Bahkan jika kita menginginkan ekspresi reguler dan FA untuk bahasa intersection, kita tidak perlu menemukan ekspresi reguler yang pergi dengan dua mesin komponen ini. Namun, ini adalah latihan yang baik dan algoritma untuk melakukan ini disajikan sebagai bagian dari bukti Teorema Kleene. Ingatlah bahwa kita melalui tahapan transisi graph dengan sisi berlabel oleh ekspresi reguler. FAI' menjadi:



State 3 adalah bagian dari tidak ada jalur dari - ke +, sehingga dapat diabaikan. Ketika kita menghilangkan sisi dari 2 ke state 1, kita mmenghilangkan loop ab pada state 1, loop bb*a pada state 2, dan jalur: state 2, state 1, +, yang menambahkan bb*-sisi dari 2 ke +.



Mungkin Jalur yang dari - ke + sekarang dari - ke state I ke +, yang diberi label (b + ab)*, dan dari - ke keadaan 1 ke keadaan 2 ke + berlabel

$$(b + ab)^*a(bb^*a)^*(\varepsilon + bb^*)$$

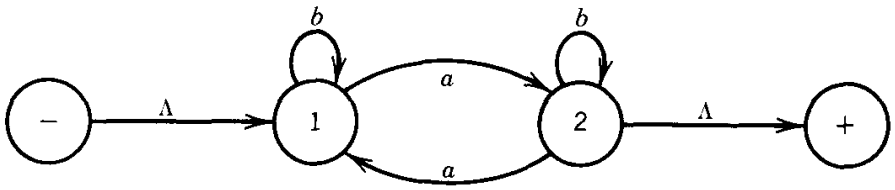
Ekspresi reguler yang telah kami hasilkan untuk L1' adalah:

$$(b + ab)^* + (b + ab)^*a(bb^*a)^*(A + b^*)$$

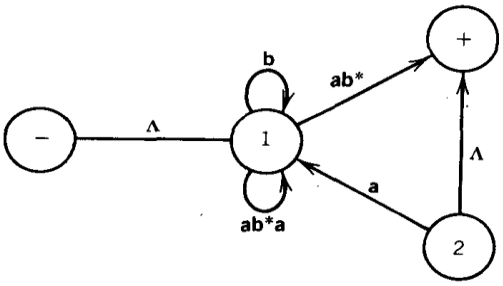
Ada banyak ekspresi reguler untuk set ini. Daripada menggunakan rumus yang rumit ini, mari kita perhatikan bahwa (b + ab)* semua kata yang tidak memiliki double a dan berakhiran b. Oleh karena itu, kami dapat menghasilkan set yang sama dengan menambahkan faktor (tidak ada atau a) ke akhir ekspresi ini:

$$rl' = (b + ab)^*(\varepsilon + a)$$

Ini adalah ekspresi reguler yang kami gunakan untuk L1'.Mari kita lakukan hal yang sama untuk bahasa L2'. FA2' menjadi:

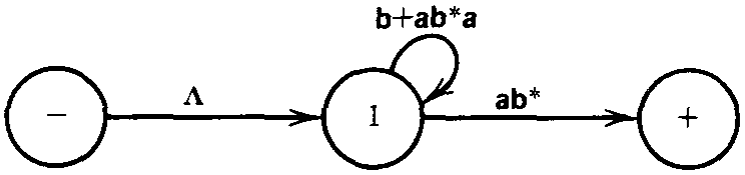


Mari kita mulai penyederhanaan gambar ini dengan menghilangkan tepi dari nyatakan 1 sampai dengan 2. Kita harus bertanya jalan apa yang rusak ini yang harus kita ganti. Satu jalur yang dihancurkannya adalah loop kembali ke state 1. Kami mengganti ini dengan loop pada state 1 berlabel ab*a, yang merupakan jumlah loop lama melalui state 2 ke. Jalur lain yang dihancurkan eliminasi ini adalah jalur dari state 1 ke state 2 untuk menyatakan +. Kita harus menyertakan tepi untuk jalur ini berlabel ab*. Gambar ini sekarang:

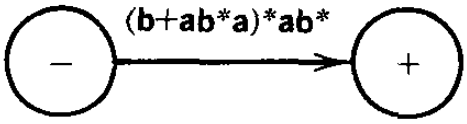


Kita sekarang dapat melihat bahwa tidak ada cara untuk mencapai state 2, jadi itu tidak mungkin bagian dari setiap jalur dari - ke +. Oleh karena itu, kita dapat menghilangkannya dan keduanya sisi. (Sebenarnya, untuk bekerja sepenuhnya sesuai dengan algoritme, kita harus hilangkan sisinya terlebih dahulu dan kemudian

perhatikan bahwa itu bisa dihapus. Kami telah mengambil kebebasan untuk mengambil jalan pintas yang berwawasan luas.) Dua putaran pada state dapat digabungkan untuk memberikan:



Kami sekarang dapat menghilangkan state I dan sehingga menjadi



ekspresi regulernya adalah r2' = (b + ab*a)*ab*

Ini adalah salah satu dari beberapa ekspresi reguler yang mendefinisikan bahasa semua kata dengan bilangan ganjil a. lainnya adalah b*ab*(ab*ab*)*

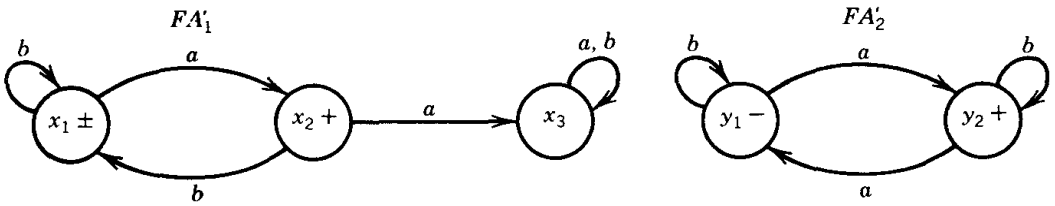
Yang kita dapatkan dengan menambahkan faktor b*a di depan ekspresi reguler untuk LI. Ini berfungsi karena kata-kata dengan angka ganjil dapat ditafsirkan sebagai b*a di depan kata-kata dengan jumlah a genap. Fakta bahwa ini dua kspresi reguler yang berbeda mendefinisikan bahasa yang sama tidak jelas. Itu pertanyaan, "Bagaimana kita bisa tahu ketika dua ekspresi reguler sama?", akan dijawab di Babberikutnya. Kami sekarang memiliki ekspresi reguler untuk LI' dan L2 ', sehingga kami dapat menulis ekspresi reguler untuk Ll' + L2'. Ini akan menjadi

$$rl' + r\,2' = (b + ab)^*(\varepsilon + a) + (b + ab^*a)^*abz$$

Kita sekarang harus pergi ke arah lain dan membuat ekspresi reguler ini menjadi sebuah FA sehingga kita dapat mengambil komplemennya untuk mendapatkan FA yang mendefinisikan L, ⋂ L 2.

Untuk membangun FA yang sesuai dengan ekspresi reguler yang rumit, seperti yang kita ingat dari bukti Teorema Kleene. Pendekatan alternatif adalah membuat mesin untuk L1' + L2' langsung dari mesin untuk L1' dan L2' tanpa menggunakan ekspresi reguler. Kita punya sudah mengembangkan metode membangun mesin yang merupakan jumlah dari dua FA juga dalam bukti Teorema Kleene.

Mari kita beri label state di dua mesin untuk FA1' dan FA2' seperti yang ditunjukkan:



di mana keadaan awal adalah x, dan yj dan keadaan akhir adalah x1, x2, dan Y2.

Enam kemungkinan keadaan kombinasi adalah:

Z1= x1 atau yl start, final (kata-kata yang diakhiri di sini diterima di FAI')

Z2 = x1 atau Y2 final (kata-kata yang diakhiri di sini diterima di FA1 ' dan FA2')

Z3 = x2 atau yj final (kata-kata yang berakhiran di sini diterima di FA1 ')

Z4 = x2 atau Y2 final (kata-kata yang diakhiri di sini diterima di FA1 ' dan FA2')

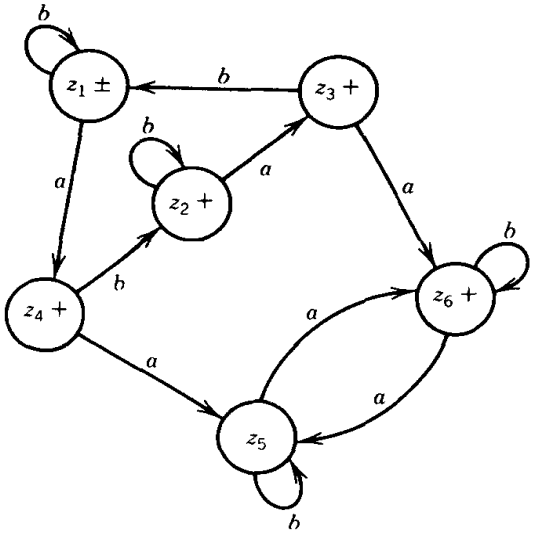
Z5 = x3 atau yj belum final

Z6 = x3 atau Y2 final (kata-kata yang berakhir di sini diterima di FA2')

Tabel transisi untuk mesin ini adalah:

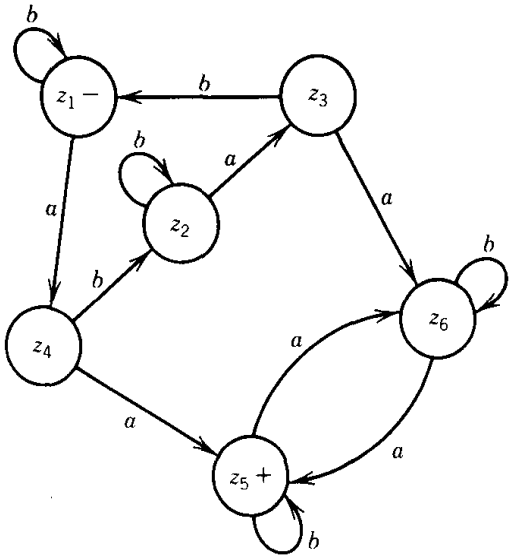
	a	*b*
$+z_1$	z_4	z_1
$+z_2$	z_3	z_2
$+z_3$	z_6	z_1
$+z_4$	z_5	z_2
z_5	z_6	z_5
$+z_6$	z_5	z_6

Dan mesinnya dapat digambarkan seperti ini:

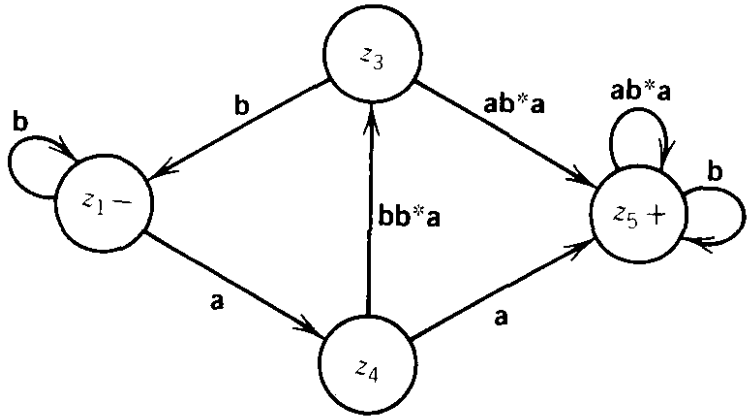


Ini adalah FA yang menerima bahasa L1' + L2'. Jika kita membalikkan state

dari setiap state dari final ke nonfinal dan sebaliknya, kami menghasilkan FA untuk bahasa L, ⋂ L2. Adalah

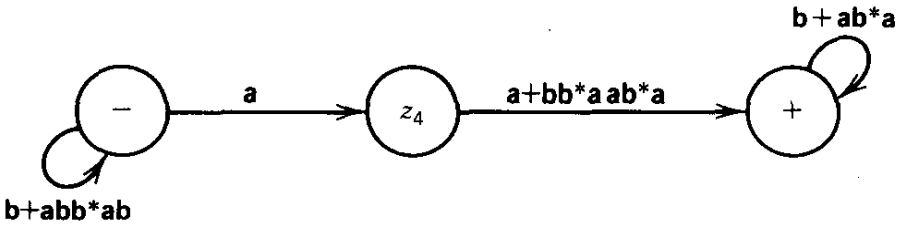


bahwa hanya ada satu - dan satu +, jadi kita tidak perlu menambahkan terminal tambahan state bagian. Mari kita mulai dengan menghapus sisi dari z2 ke z3. Seluruh jalan dari z4 hingga z2 hingga z3 dapat diganti dengan sisi berlabel bb*a. Mari kita juga menghilangkan sisi dari z3 ke z6 dan biarkan z3 langsung ke + pada sisi berlabel ab*a. Tidak perlu menyertakan di jalur z3 kemungkinan perulangan bolak-balik antara z6 dan + yang dapat dilakukan setelah z3 mengarah ke +. Begitu kita sampai +, kita bisa melompat keluar dan kembali lagi. Kami kemudian dapat mengganti z6 dengan loop di + berlabel ab*a. Gambarnya sekarang menjadi



Sekarang ada dua jalur yang dimulai dari - dan kembali ke -, yang sederhana loop di - dan sirkuit - ke z4 ke z3 ke -. Jalur kedua dapat dikurangi ke loop berlabel (abb*ab), dan kemudian kita dapat menghapus tepi dari z3 ke -. Dari z4 ke + ada dua jalur: satu melalui z3 dan yang lainnya langsung.

Kita dapat menghapus sisi Z4 ke Z3 (dan kemudian seluruh status z3) dengan memberi label sisi langsung z4 ke + dengan kedua alternatif seperti yang ditunjukkan di bawah ini:



Seluruh mesin ini direduksi menjadi ekspresi reguler:

$$(b + abb^*ab)^*a(a + bb^*aab^*a)(b + ab^*a)^*$$

Padahal kita tahu ungkapan ini pasti jawaban kita karena kita tahu bagaimana itu diturunkan, mari kita coba menganalisisnya untuk melihat apakah kita bisa mengerti apa arti bahasa ini .dalam pengertian yang lebih intuitif.

Seperti berdiri, ada empat faktor (yang kedua hanya a dan yang pertama dan keempat berbintang). Setiap kali kami menggunakan salah satu opsi dari keduanya faktor akhir kami memasukkan jumlah a yang genap ke dalam kata (baik tidak ada atau dua). Faktor kedua memberi kita bilangan ganjil dari a (tepat satu). Itu faktor ketiga memberi kita pilihan untuk mengambil satu atau tiga a. Secara total, jumlah a harus genap. Jadi semua kata dalam bahasa ini ada di L2.

Faktor kedua memberi kita a, dan kemudian kita harus segera menggabungkannya ini dengan salah satu pilihan dari faktor ketiga. Jika kita memilih a, maka kami telah membentuk ganda a. Jika kita memilih ekspresi lain, bb*aab*a, maka kita telah membentuk double a dengan cara yang berbeda. Dengan salah satu pilihan kata-kata dalam bahasa ini semuanya memiliki a ganda dan oleh karena itu dalam L1.

Ini berarti bahwa semua kata dalam bahasa ekspresi reguler ini terkandung dalam bahasa L1 ∩ L2 . Tetapi apakah semua kata dalam L1∩L2 termasuk? dalam bahasa ungkapan ini? Jawaban untuk ini adalah ya. Mari kita lihat

kata apa saja yang ada di L,⋂ L 2. Itu memiliki jumlah a yang genap dan a ganda di suatu tempat di dalamnya. Ada dua kemungkinan untuk dipertimbangkan secara terpisah:

1. Sebelum dobel pertama a ada bilangan genap dari a.

2. Sebelum dobel pertama a ada bilangan ganjil dari a.

Kata-kata tipe 1 berasal dari ungkapan di bawah ini.

(jumlah a genap tetapi tidak digandakan) (aa pertama)

(jumlah a genap dapat digandakan)

= (b + abb*ab)* (aa)(b + ab*a)*

= tipe 1.

Perhatikan bahwa faktor ketiga mendefinisikan bahasa L, dan merupakan ekspresi yang lebih pendek dari r, kami gunakan di atas.

Kata-kata tipe 2 berasal dari ungkapan:

(angka ganjil dari a yang tidak digandakan) (aa pertama)

(angka ganjil dari a dapat digandakan)

Perhatikan bahwa faktor pertama harus diakhiri dengan b, karena tidak ada a yang merupakan bagian dari aganda a.

= [(b + abb*ab)*abb*] aa [b*a(b + ab*a)*]

= (b + abb*ab)*(a)(bb* aab*a)(b + ab*a)*

- tipe 2

Menjumlahkan tipe I dan tipe 2 bersama-sama (dan memfaktorkan suku-suku sejenis menggunakan hukum distributif), kami memperoleh ekspresi yang sama yang kami dapatkan dari algoritma. Kami sekarang memiliki dua bukti bahwa ini memang ekspresi reguler untuk bahasa L1 ⋂ L2.

Ini melengkapi perhitungan. Itu pertama dibuktikan dengan ekspresi reguler dan TG, yang kedua dengan FA, dan ketiga dengan diagram Venn.

Kita harus mengakui sekarang bahwa bukti teorema bahwa persimpangan dari dua bahasa reguler lagi-lagi bahasa reguler adalah pedagogis yang jahat menipu. Teorema ini sebenarnya tidak sesulit yang kita bayangkan. Kami memilih cara yang sulit untuk melakukan sesuatu karena itu adalah contoh matematika yang bagus berpikir: Kurangi masalah menjadi elemen-elemen yang telah dipecahkan.

Prosedur ini mengingatkan pada cerita terkenal tentang ahli matematika teoretis. Profesor X terkejut suatu hari menemukan mejanya terbakar. Dia berlari

ke alat pemadam dan memadamkan api. Keesokan harinya dia mendongak dari buku untuk melihat bahwa keranjang sampahnya terbakar. Dengan cepat dia mengambil keranjangnya dan mengosongkannya ke mejanya yang mulai terbakar. Dengan demikian mengurangi masalah yang sudah dia selesaikan, dia kembali ke bacaannya. (Itu siswa yang menganggap ini lucu mungkin adalah orang-orang yang telah mengatur kebakaran di kantornya.)

Berikut ini adalah bukti yang lebih langsung bahwa perpotongan dua bahasa regular bahasa adalah regular.

Bukti Teorema

Mari kita ingat kembali metode yang kita perkenalkan sebagai bagian dari membuktian Teorema Kleene untuk menunjukkan bahwa untuk setiap FAI dan FA2 ada FA3 yang menerima bahasa tersebut itu adalah gabungan dari dua bahasa ini, yaitu, FA3 menerima string apa pun yang diterima oleh FA1 atau FA2.

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

Untuk membuktikannya, kami menunjukkan cara membuat mesin dengan state z1, z2, ... dari bentuk X jika input berjalan di FA1 atau Y jika inputnya berjalan di FA2. Jika salah satu state x atau state y adalah state akhir, kami membuat z menyatakan state akhir.

Mari kita sekarang membangun mesin FA3 yang sama persis tetapi mari kita ubah penunjukannya state akhir. Biarkan state z menjadi state akhir hanya jika keduanya sesuai state x dan state y yang sesuai adalah state akhir. Sekarang FA3 hanya menerima string yang mencapai state akhir secara bersamaan di kedua mesin. Kata-kata dalam bahasa untuk FA3 adalah kata-kata dalam kedua bahasa untuk FAI dan FA2. Oleh karena itu, ini adalah mesin untuk bahasa dari intersection. Tidak hanya buktinya tetapi juga konstruksi mesinnya.

Contoh

Dalam pembuktian Teorema Kleene, kita ambil jumlah mesin yang menerima kata-kata dengan ganda a

	a	b
$-x_1$	x_2	x_1
x_2	x_3	x_1
$+x_3$	x_3	x_3

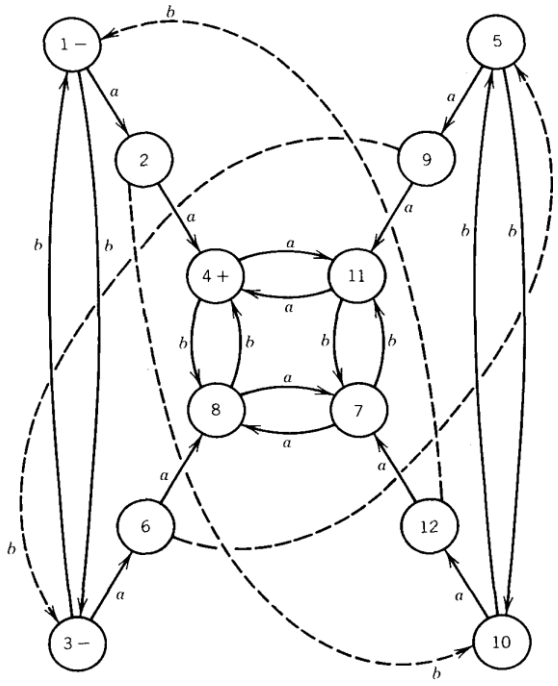
dan mesin yang menerima semua kata

	a	b
$\pm y_1$	y_3	y_2
y_2	y_4	y_1
y_3	y_1	y_4
y_4	y_2	y_3

Mesin yang dihasilkan adalah

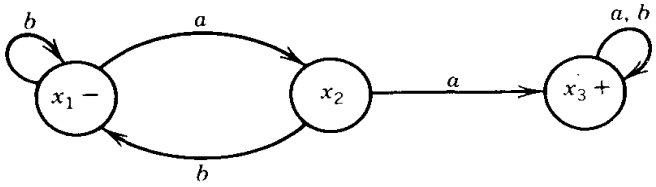
	a	b	Old States
$\pm z_1$	z_2	z_3	x_1 or y_1
z_2	z_4	z_5	x_2 or y_3
z_3	z_6	z_1	x_1 or y_2
$+z_4$	z_7	z_8	x_3 or y_1
z_5	z_9	z_{10}	x_1 or y_4
z_6	z_8	z_{10}	x_2 or y_4
$+z_7$	z_4	z_{11}	x_3 or y_3
$+z_8$	z_{11}	z_4	x_3 or y_2
z_9	z_{11}	z_1	x_2 or y_2
z_{10}	z_{12}	z_5	x_1 or y_3
$+z_{11}$	z_8	z_7	x_3 or y_4
$+z_{12}$	z_7	z_3	x_2 or y_1

Mesin intersection identik dengan ini kecuali hanya memiliki satu final state. Agar state z menjadi state akhir, state x dan y harus menjadi state akhir. Jika FAI dan FA2 hanya memiliki satu state akhir, maka FA3 dapat memiliki hanya satu state akhir (jika dapat dicapai sama sekali). Satu-satunya state terakhir adalah FA3 adalah z4, yaitu (x3 atau y1). Mesin yang rumit ini digambarkan di bawah ini:

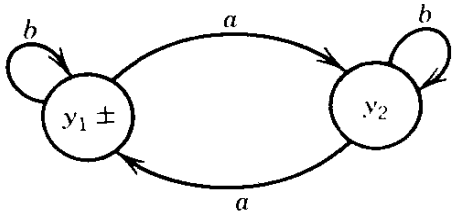


Garis putus-putus adalah sisi yang sangat bagus, tetapi harus bersilangan dengan sisi yang lain. Dengan sedikit imajinasi, kita bisa melihat bagaimana mesin ini menerima semuanya BAHKAN dengan dobel a. Semua perubahan disebabkan oleh b, dengan a. Untuk masuk ke dalam empat state mengambil a ganda.

Contoh FA1 dengan L = semua string dengan min a ganda menuju final state



FA2 dengan L= semua string dengan jumlah a genap

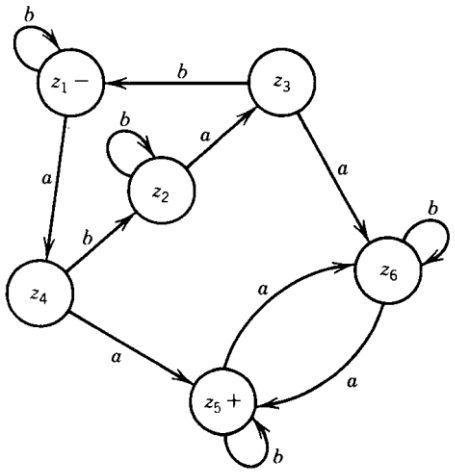


Mesin yang mensimulasikan input yang sama yang berjalan pada kedua mesin di sekali adalah:

	a	b	Old States
$-z_1$	z_4	z_1	x_1 or y_1
z_2	z_3	z_2	x_1 or y_2
z_3	z_6	z_1	x_2 or y_1
z_4	z_5	z_2	x_2 or y_2
z_5	z_6	z_5	x_3 or y_1
z_6	z_5	z_6	x_3 or y_2

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

Agar dapat diterima oleh FA1, string input harus memiliki jalur yang diakhiri dengan x3 . Ke diterima oleh FA2, string input harus memiliki jalur yang diakhiri dengan yl. Untuk diterima oleh kedua mesin sekaligus, string input pada mesin-z, mulai pemrosesannya dalam z1 , harus mengakhiri jalurnya dalam state z5 dan hanya z5.



Contoh

Mari kita bekerja melalui satu contoh terakhir dari persimpangan. Dua bahasa kami akan menjadi

L•= semua kata yang dimulai dengan a

L2= semua kata yang berakhiran a

rl= a (a + b)*

= (a + b)*aa

Bahasa intersectonya menjadi L1 ⋂ L2 = semua kata yang diawali dan diakhiri dengan huruf a

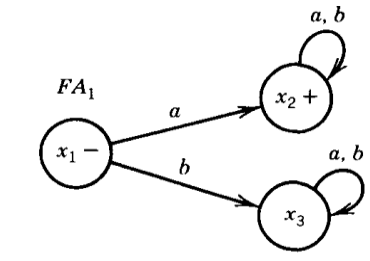
Bahasanya jelas reguler karena dapat didefinisikan oleh reguler ekspresi

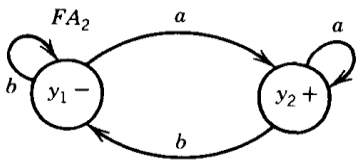
a(a + b)*a + a

Perhatikan bahwa suku pertama mengharuskan a pertama dan terakhir berbeda, yang: itulah mengapa kita membutuhkan pilihan kedua "+ a."

Dalam contoh ini kami cukup beruntung untuk "memahami" bahasa, jadi kita bisa mengarang ekspresi reguler yang kita "pahami" mewakili persimpangan. Secara umum, ini tidak terjadi, jadi kami mengikuti algoritme yang disajikan dalam bukti, yang dapat kami jalankan bahkan tanpa memahami manfaat.

Untuk ini kita harus mulai dengan FA yang mendefinisikan bahasa-bahasa ini:

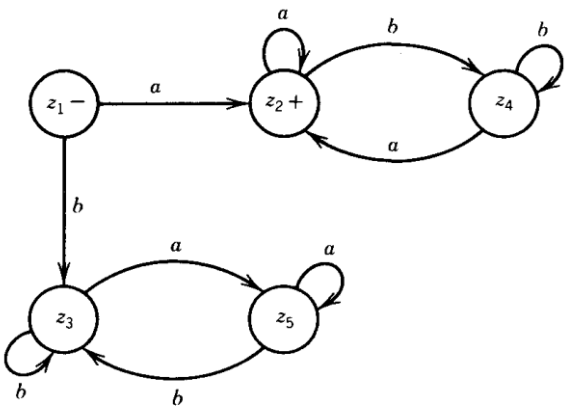




Ternyata, meskipun dua ekspresi reguler sangat mirip, mesin sangat berbeda. Ada FA2 versi tiga state tetapi tidak ada FA1 versi dua state. Kami sekarang membangun tabel transisi dari mesin yang menjalankan string inputnya pada FA1 dan FA2 secara bersamaan.

State	Read a	Read b	New names
x_1 or y_1	x_2 or y_2	x_3 or y_1	$-z_1$
x_2 or y_2	x_2 or y_2	x_2 or y_1	z_2
x_3 or y_1	x_3 or y_2	x_3 or y_1	z_3
x_2 or y_1	x_2 or y_2	x_2 or y_1	z_4
x_3 or y_2	x_3 or y_2	x_3 or y_1	z_5

Mesinnya terlihat seperti ini:



8.6 Latihan Soal

Jika L1 dan L2 adalah bahasa regular yang didefinikanan dengan regular ekspresi dibawah ini, tentukan L1+L2. L1L2 serta L2* juga merupakan bahasa regular dengan pembuktian regular ekpresi maupun dengan mesin automata.

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

	L_1	L_2
1.	**(a + b)*a**	**b(a + b)***
2.	**(a + b)*a**	**(a + b)*aa(a + b)***
3.	**(a + b)*a**	**(a + b)*b**
4.	**(a + b)b(a + b)***	**b(a + b)***
5.	**(a + b)b(a + b)***	**(a + b)*aa(a + b)***
6.	**(a + b)b(a + b)***	**(a + b)*b**
7.	**(b + ab)*(a + Λ)**	**(a + b)*aa(a + b)***
8.	**(b + ab)*(a + Λ)**	**(b + ab*a)*ab***
9.	**(b + ab)*(a + Λ)**	**(a + ba)*a**
10.	**(ab*)***	**b(a + b)***

Bab 9

Context-Free Grammars

9.1 Context-Free Grammars

Tujuan keseluruhan kami adalah mempelajari komputer: Apa itu? dari apa mereka? tersusun? Apa yang bisa mereka lakukan dan apa yang tidak bisa mereka lakukan? Apa yang akan mereka dapat? lakukan dalam sepuluh ribu tahun? Mesin-mesin Bagian I tidak bisa berbuat banyak. Ini adalah karena mereka tidak memiliki cukup komponen. Kami akan segera memperbaiki ini kekurangan. Di bab ini kita akan menghubungkan secara langsung dengan beberapa dari masalah dalam Ilmu Komputer yang sudah kita kenal. Komputer paling awal, seperti kalkulator tangan yang dapat diprogram dari 1970-an, tidak menerima instruksi kecuali dalam bahasa mesin mereka sendiri atau (hampir setara) dalam bahasa rakitan mereka sendiri. Setiap prosedur betapapun rumitnya, harus dijabarkan dalam rangkaian instruksi yang paling kasar: Hal ini diperlukan lusinan instruksi primitif untuk melakukan sesuatu yang berguna. Secara khusus, dibutuhkan beberapa waktu untuk mengevaluasi satu aritmatika yang ekspresi rumit.

9.2 Tata bahasa

Untuk mempelajari bahasa secara matematis, kita membutuhkan mekanisme untuk mendeskripsikannya. Bahasa sehari-hari tidak tepat dan ambigu, sehingga deskripsi informal dalam Bahasa sering tidak memadai. Notasi himpunan yang digunakan sebagai contoh lebih cocok, tetapi terbatas. Saat kita melanjutkan, kita akan belajar tentang beberapa mekanisme definisi bahasa yang berguna dalam situasi yang berbeda. Di sini kami memperkenalkan yang umum dan kuat, gagasan tentang tata bahasa.

Tata bahasa untuk bahasa Inggris memberi tahu kita apakah kalimat tertentu terbentuk dengan baik atau tidak. Aturan khas tata bahasa Inggris adalah "sebuah kalimat" dapat terdiri dari frase kata benda yang diikuti oleh predikat." Lebih ringkasnya kita tulis ini sebagai kalimat → frase kata benda predikat, dengan interpretasi yang jelas. Ini, tentu saja, tidak cukup untuk dihadapi kalimat yang sebenarnya. Kita sekarang harus memberikan definisi untuk yang baru diperkenalkan membangun frase kata benda dan predikat . Jika kita melakukannya dengan frase nomina → artikel kata benda , predikat → kata kerja , dan jika kita mengasosiasikan kata sebenarnya "a" dan "the" dengan artikel , "anak laki-laki" dan "anjing" dengan kata benda , dan "berlari" dan "berjalan" dengan kata kerja , maka tata bahasanya memberi tahu kita bahwa kalimat "anak laki-laki berlari" dan "anjing berjalan" benar terbentuk. Jika kita harus memberikan tata bahasa yang lengkap, maka secara teori, setiap hak kalimat dapat dijelaskan seperti ini. Contoh ini menggambarkan definisi konsep umum dalam istilah dari yang sederhana. Kami mulai dengan konsep tingkat atas, di sini kalimat , dan berturut-turut menguranginya menjadi blok bangunan bahasa yang tidak dapat direduksi. Itu generalisasi ide-ide ini membawa kita ke tata bahasa formal.

Mari kita kembali ke analogi antara bahasa komputer dan bahasa Inggris. Beberapa aturan tata bahasa Inggris adalah sebagai berikut:

a. a sentence can be a subject followed by a predicate.
b. a subject can be a noun-phrase.
c. a noun-phrase can be an adjective followed by a noun-phrase.
d. a noun-phrase can be an article followed by a noun-phrase.
e. a noun-phrase can be a noun.

f. a predicate can be a verb followed by a noun-phrase.
g. a noun can be:
   apple bear cat dog
h. a verb can be:
   Eats follows gets hugs
i. An adjective can be:
   Itchy jumpy
j. An article can be:
   a an the

Mari kita, untuk saat ini, kita membentuk kalimat dengan Aturan-aturan yang disebutkan di atas. Dalam aturan bahasa Inggris ini ada ratusan kalimat yang bisa kita bentuk. Sebagai contoh:

The itchy bear hugs the jumpy dog

Tabel 9.1: Metode bagaimana kalimat ini dapat dihasilkan diuraikan di bawah ini:

1	sentence --->	sentence => subject predict	Rule 1
2	--->	noun-phrase predicate	Rule 2
3	--->	noun-phrase verb noun-phrase	Rule 6
4	--->	article noun-phrase verb noun-phrase	Rule 4
5	--->	article adjective noun-phrase verb noun-phrase	Rule 3
6	--->	article adjective noun verb noun-phrase	Rule 5
7	--->	article adjective noun verb article noun-phrase	Rule 4
8	--->	article adjective noun verb article adjective noun-phrase	Rule 3
9	--->	article adjective noun verb article adjective noun	Rule 5
10	--->	the adjective noun verb article adjective noun	Rule 10
11	--->	the itchy noun verb article adjective noun	Rule 9
12	--->	the itchy bear verb article adjective noun	Rule 7
13	--->	the itchy bear hugs article adjective noun	Rule 8
14	--->	the itchy bear hugs the adjective noun	Rule 10
15	--->	the itchy bear hugs the jumpy noun	Rule 9
16	--->	the itchy bear hugs the jumpy dog	Rule 7

Kita dapat mengikuti model yang sama untuk mendefinisikan ekspresi aritmatika. Kita dapat tulis seluruh sistem aturan pembentukan sebagai daftar kemungkinan substitusi ditunjukkan di bawah ini

Start ---> AE
AE ---> (AE + AE)
AE ---> (AE - AE)
AE ---> (AE *AE)
AE ---> (AE AE)
AE ---> (AE ** AE)
AE ---> (AE)
AE ---> - (AE)
AE ---> ANY-NUMBER

Di sini kami telah menggunakan kata "strat" untuk memulai proses, seperti yang kami gunakan kata "Kalimat" dalam contoh bahasa Inggris. Selain Start, satu-satunya yang lain nonterminal adalah AE. Terminalnya adalah frasa "nomor apa saja" dan simbolnya

+ - * /** ( )

kita dapat mendefinisikan frasa ini dengan seperangkat aturan, sehingga mengonversi dari terminal menjadi nonterminal.

Rule 1	NY-NUMBER	--->	FIRST-DIGIT
Rule 2	FIRST-DIGIT	--->	FIRST DIGIT OTHER-DIGIT
Rule 3	FIRST-DIGIT	--->	123456789
Rule 4	OTHER-DIGIT	--->	0 123456789

turan 3 dan 4 menawarkan pilihan terminal. Kami memberi spasi di antara mereka untuk menunjukkan "pilih satu," tapi kami akan segera memperkenalkan simbol disjungtif lain. Kami dapat memproduksi nomor 1066 sebagai berikut:

ANY-NUMBER	--->	FIRST-DIGIT
Rule 2	--->	FIRST-DIGIT OTHER-DIGIT
Rule 2	--->	FIRST-DIGIT OTHER-DIGIT OTHER-DIGIT
Rule 2	--->	FIRST-DIGIT OTHER-DIGIT OTHER-DIGIT OTHER-DIGIT
Rule 3 dan 4	--->	1066

Beberapa aturan tata bahasa melibatkan terminal dan nonterminal bersama-sama, menggunakan aturan sebagai berikut

Satu Nonterminal ---> string dari Nonterminal-Nonterminal, atau

Satu Nonterminal ---> pilihan terminal (T)

Definisi

Tata bahasa bebas konteks, yang disebut CFG, adalah kumpulan dari tiga hal:

1. Dari Alfabet ∑ yang dapat disebut terminal yang akan terbentuk string menjadi kata-kata dari suatu bahasa.
2. Himpunan simbol yang disebut nonterminal, salah satunya adalah simbol awal S, untuk "mulainya di sini."
3. Seperangkat bentuk produksi yang terbatas

Atau dapat juga didefinisikan sebagai Grammer (G) = (V,T, S, P) , di mana V adalah himpunan berhingga dari objek yang disebut variabel, T adalah himpunan berhingga dari objek yang disebut simbol terminal, S adalah simbol khusus yang disebut variabel awal, P adalah himpunan berhingga dari aturan produksi. Diasumsikan tanpa menyebutkan lebih lanjut bahwa himpunan V dan T tidak kosong dan terputus-putus.

Aturan produksi sebuah tata bahasa yang menentukan bagaimana tata bahasa mengubah satu string menjadi string lain, dan melalui ini mereka mendefinisikan bahasa yang berhubungan dengan tata bahasa. sebuah produksi dapat digunakan kapanpun itu berlaku, dan itu dapat diterapkan sesuai keinginan. Jika w1 w2 • • •⇒wn, kita katakan bahwa w1 menurunkan wn dan menulis w1 wn. menunjukkan bahwa jumlah langkah yang tidak ditentukan (termasuk nol) dapat menjadi diambil untuk menurunkan wn dari w1. Dengan menerapkan aturan produksi dalam urutan yang berbeda, tata bahasa yang diberikan biasanya dapat menghasilkan banyak string. Himpunan semua string terminal tersebut adalah bahasa yang ditentukan atau dihasilkan oleh tata bahasa.

Contoh,

1. CFG

Aturan produksi 1 S ---> aS

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

Aturan produksi 2 S ---> Ɛ

Jika menerapkan Produksi 1 enam kali dan kemudian menerapkan Produksi 2, maka menghasilkan,

S ---> Ɛ

S ---> aS
s --> a Ɛ
S --> a

S ---> aS
---> aaS
---> aa Ɛ
---> aa

S ---> aS
---> aaS
---> aaaS
---> aaa Ɛ
---> aaa

S ---> aS
---> aaS
---> aaaS
---> aaaaS
---> aaaa Ɛ
---> aaaa

S ---> aS
---> aaS
---> aaaS
---> aaaaS
---> aaaaaƐ
---> aaaaa

S ---> aS
---> aaS
---> aaaS
---> aaaaS

---> aaaaaS

---> aaaaaaS

---> aaaaaaƐ

---> aaaaaa

L (G) = {Ɛ, a, aa, aaa, aaaa, aaaaa, aaaaaa, ……., ……., }

2. CFG;

S ---> SS aturan 1

S ---> a aturan 2

S ---> Ɛ aturan 3

Dalam tata bahasa ini kita dapat menghasilkan derivasi berikut.

S ---> SS

S ---> SSS

S ---> SaS

S ---> SaSS

S ---> Ɛ aSS

S ---> Ɛ aaS

S ---> Ɛ aa Ɛ

S ---> aa

L (G) = {Ɛ, a, aa, aaa, aaaa, aaaaa, aaaaaa, ……., …….,….},

Bukti

1 S ---> Ɛ 3

2 S ---> a 2

3 S ---> SS 1

---> aS 2

---> aa 2

4 S ---> SS 1

---> SSS 1

---> aSS 2

---> aaS 2

---> aaa 2

5 S ---> SS 1

---> SSS 2

---> SSSS 2

---> aSSS 2

---> aaSS 2

		--->	aaaS	2
		--->	aaaa	2
6	S	--->	SS	
		--->	SSS	
		--->	SSSS	
		--->	SSSSS	
		--->	aSSSS	
		--->	aaSSS	
		--->	aaaSS	
		--->	aaaaS	
		--->	aaaaa	
7	S	--->	SS	
		--->	SSS	
		--->	SSSS	
		--->	SSSSS	
		--->	SSSSSS	
		--->	aSSSSS	
		--->	aaSSSS	
		--->	aaaSSS	
		--->	aaaaSS	
		--->	aaaaaS	
		--->	aaaaaa	

Contoh 3 CFG,

S	--->	aS
S	--->	bS
S	--->	a
S	--->	b

Menghasilkan kata baab sebagai berikut:

S	--->	bS
	--->	baS
	--->	baaS
	--->	baab

Contoh 4, CFG

S	--->	aS

S ---> bS

S ---> a

S ---> b

S ---> Ɛ

Menghasilkan kata baab sebagai berikut:

S ---> aS

---> abS

---> abƐ

---> ab

Contoh CFG, dimana Terminal a dan b dan nonterminalnya adalah S, A, dan B.

S ---> aB

S ---> bA

A ---> a

A ---> aS

A ---> bAA

S ---> aS

B ---> b

B ---> bS

B ---> aBB

Bahasa yang dihasilkan CFG ini adalah bahasa EQUAL dari semua string

yang memiliki jumlah a dan b yang sama di dalamnya. Bahasa ini dimulai

EQUAL = {ab, ba, aabb, abab, abba, baab, baba, bbaa, aaabbb,..., …. }

Kami sekarang memperkenalkan simbol, 1, garis vertikal, yang berarti disjungsi (atau). Dengan menggabungkan semua produksi yang memiliki sisi kiri yang sama. Sebagai contoh,

S ---> aS

S ---> Ɛ

Dapat ditulis,

S ---> aS | Ɛ

CFG

S ---> X

S ---> Y

X ---> Ɛ

Y ---> aY

Y ---> bY

Y ---> a

Y ---> b

menjadi

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

S ---> X | Y

X ---> Ɛ

Y ---> aY | bY | a | b

Contoh

Biarkan terminal menjadi a dan b, biarkan nonterminal menjadi S dan X, dan biarkan produksi menjadi

S → XaaX

X→ aX

X→ bX

X→A

Kita sudah tahu dari contoh sebelumnya bahwa tiga produksi terakhir akan memungkinkan kita untuk menghasilkan kata apa pun yang kita inginkan dari nonterminal X. Jika nonterminal X muncul di string kerja apa pun yang dapat kita terapkan produksinya menjadi kata yang kita inginkan. Oleh karena itu, kata-kata yang dihasilkan dari S memiliki membentuk apapun aa apapun atau

$$(a + b)^*aa(a + b)^*$$

yang merupakan bahasa semua kata dengan double a di suatu tempat. Misalnya, untuk menghasilkan baabaab kita dapat melanjutkan sebagai berikut:

S → XaaX → bXaaX → baXaaX → baaXaaX → baabXaaX

→ baabƐaaX = baabaaX → baabaabX → baabaabƐ = baabaab

Ada urutan lain yang juga menurunkan kata baabaab.

Contoh

Biarkan terminal menjadi a dan b, biarkan nonterminal menjadi S, X, dan Y dan biarkan produksi menjadi

S → XY

X → aX

X → bX

X → a

Y → Ya

Y → Yb

Y → a

Apa yang bisa diturunkan dari X? Mari kita lihat produksi X saja.

X→ aX

X→ bX

X→ a

Dimulai dengan X nonterminal dan memulai derivasi menggunakan dua yang pertama produksi kami selalu menyimpan X nonterminal di ujung kanan. Untuk menyingkirkan X untuk selamanya kita akhirnya harus menggantinya dengan a pada produksi ketiga. Kita dapat melihat bahwa string terminal yang berasal dari X harus diakhiri dengan a dan setiap kata yang berakhiran a dapat diturunkan dari X. Misalnya, menurunkan kata babba dari X kita dapat melanjutkan sebagai berikut:

X → bX → baX → babX→ babbX → babba

Demikian pula, kata-kata yang dapat diturunkan dari Y persis yang dimulai dengan sebuah. Untuk menurunkan abbab, misalnya, kita dapat melanjutkan:

$$Y \rightarrow Yb \rightarrow Yab \rightarrow Ybab \rightarrow Ybbab \rightarrow abbab$$

A Y selalu berada di ujung kiri sampai digantikan oleh a. Ketika sebuah Bagian X digabungkan dengan bagian Y, a ganda terbentuk.

Kita dapat menyimpulkan bahwa mulai dari S kita hanya dapat menurunkan kata-kata dengan a ganda, dan semua kata ini dapat diturunkan.

Misalnya, untuk menurunkan babaabb kita tahu bahwa bagian X harus berakhir di pertama a dari ganda a dan bahwa bagian Y harus dimulai dengan yang kedua a.

$$S \rightarrow XY \rightarrow bXY \rightarrow baXY \rightarrow babXY \rightarrow babaY$$

$$\rightarrow babaYb \rightarrow babaYbb \rightarrow babaab$$

CONTOH

Biarkan terminal menjadi a dan b. Biarkan tiga nonterminal menjadi S, BALANCED, dan TIDAK SEIMBANG. Kami memperlakukan nonterminal ini seolah-olah masing-masing adalah satu simbol dan tidak ada yang lebih membingungkan. Biarkan produksi menjadi

S→ SS

S → BALANCED S

S → S BALANCED

S→ Ɛ

S → tidak seimbang S tidak seimbang

SEIMBANG → aa

SEIMBANG → bb

TIDAK SEIMBANG → ab

tidak seimbang → ba

Kami akan menunjukkan bahwa bahasa yang dihasilkan dari produksi ini adalah himpunan dari semua kata dengan jumlah a genap dan jumlah b genap. Ini adalah teman lama kita.

Untuk membuktikan ini kita harus menunjukkan dua hal: bahwa semua kata di BAHKAN-GENAP dapat dihasilkan dari produksi ini dan bahwa setiap kata yang dihasilkan dari produksi ini sebenarnya dalam bahasa.

Pertama kita tunjukkan bahwa setiap kata dalam BAHKAN-BAHKAN dapat dihasilkan oleh ini produksi. Dari pembahasan kita sebelumnya tentang bahasa BAHKAN-BAHKAN kita ketahuilah bahwa setiap kata dalam bahasa ini dapat ditulis sebagai kumpulan substring dari

Type aa atau type bb

atau type (ab + ba) (aa + bb)* (ab + ba).

Ketiga jenis dapat dihasilkan dari S nonterminal dari produksi di atas. Berbagai substring dapat disatukan dengan aplikasi berulang dari produksi

S→SS

Produksi ini sangat berguna. Jika kita menerapkannya empat kali, kita dapat mengubah satu S menjadi lima S. Masing-masing dari S ini dapat menjadi suku kata dari salah satu dari tiga jenis. Misalnya, kata BAHKAN-GENAP aababbab dapat dihasilkan.

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

9.3 Latihan Soal

1. Ditunjukkan CFG, sebagai berikut

$$S \rightarrow aS \mid bb$$

   a. Buktikan bahwa cfg diatas dapat membangkitkan bahasa L
   b. Definisikan dengan Ekpresi regular

2. Ditunjukkan CFG

$$S \rightarrow XYX$$
$$X \rightarrow aX \mid bX \mid \Lambda$$
$$Y \rightarrow bbb$$

   a. Tentukan dengan bahasa L
   b. Tentukan dengan ekpresi regular.

Bab 10

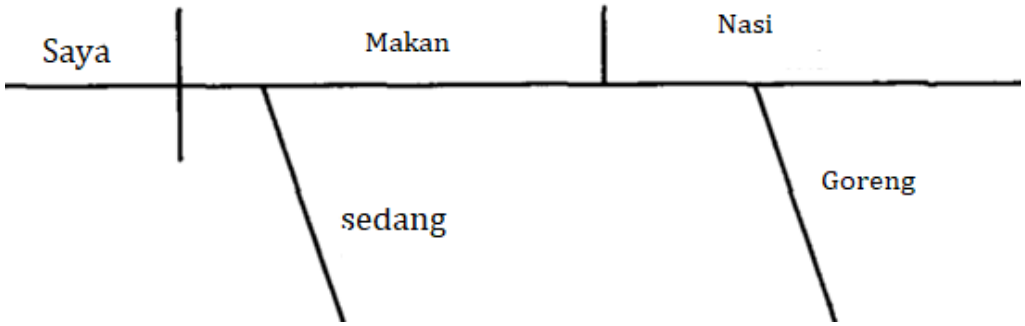
Teori Pohon

10.1 Teori Pohon

Tata bahasa adalah jenis kaidah bahasa yang mengatur kriteria penggunaan kata dan kalimat. Kedudukan kajian tata bahasa merupakan yang utama dalam pembelajaran bahasa, jika seorang siswa diminta untuk membuat sebuah pohon bahasa yang sesuai dengan aturan bahasa dan kedudukan bahasanya, maka siswa iku mungkin akan membuat pohon sesuai dengan aturan tatabahasa yang berlaku, sebagai contoh membuat sebuah pohon bahasa atau kalimat,

"Saya sedang makan nasi goreng"

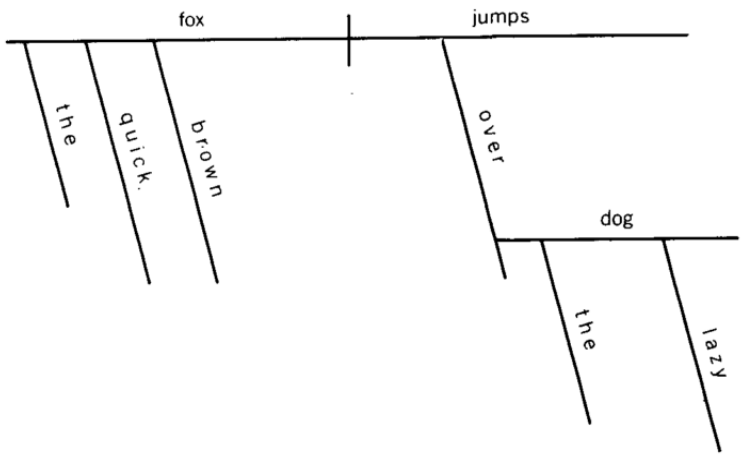
Maka pohon urai itu adalah



Gambar 10.1: Pohon Urai

Demikian dalam bahasa inggis jika harus menggambar pohon pengurai, yang merupakan gambar dengan garis dasar dibagi menjadi subjek dan predikat sebagaimana aturan yang berlaku di tatabahasa inggris, yaitu dengan semua kata atau frasa dimodifikasi dengan gambar pohon sebagai pelengkap pada garis penghubung. Sebagai contoh

The quick brown fox jumps over the lazy dog.

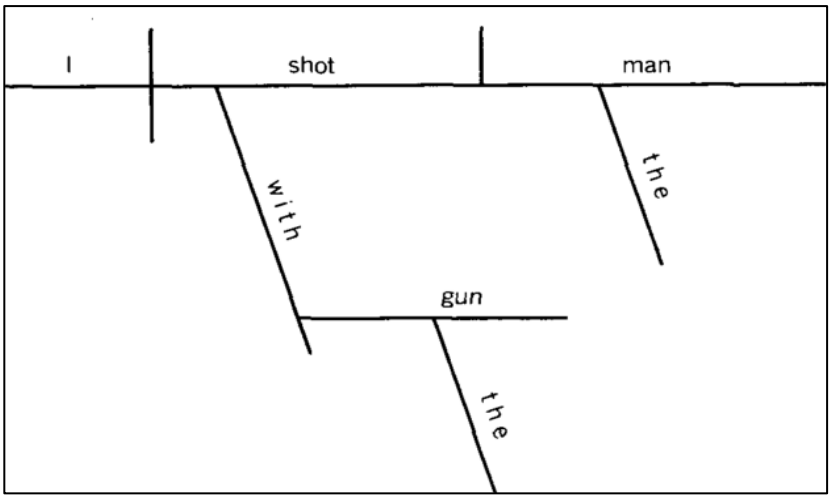


Gambar 10.2: Pohon Urai Modifikasi

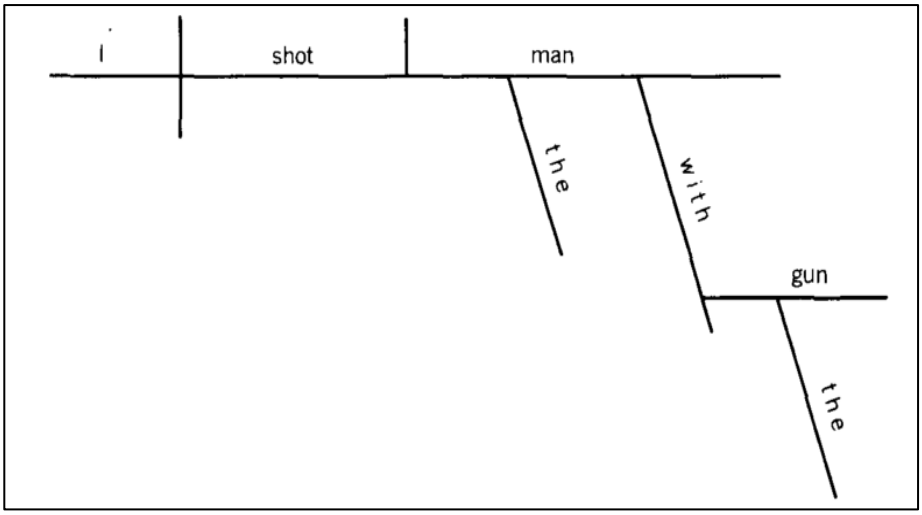
Contoh gambar pohon urai Kalimat bahasa inggris,

"I shot the man with the gun."

Diagramnya sebagaimana berikut,



Atau



Gambar 10.3: Pohon Urai Kalimat Bahasa Inggris

Dalam diagram pertama "dengan pistol" menjelaskan bagaimana saya menembak. Di detik diagram "dengan pistol" menjelaskan siapa yang saya tembak.

Diagram ini membantu kita meluruskan ambiguitas. Mereka mengubah serangkaian kata menjadi ide yang dapat ditafsirkan dengan mengidentifikasi siapa melakukan apa kepada siapa

Gagasan untuk membuat diagram sebuah kalimat untuk menunjukkan bagaimana kalimat itu harus diurai membawa lebih mudah untuk dengan aturan produksi-produksi dari tatabahasa bebas kontek CFG yang membangkitkan bahasa yang sesuan dengan grammer (G)

G = (V,T, P, S). Pohon urai untuk G adalah pohon dengan kondisi berikut:

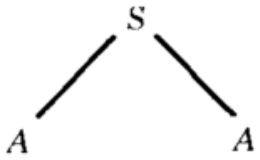
1. Setiap node diberi label oleh variabel di V.

2. Setiap daun diberi label oleh salah satu variabel, terminal, atau Ɛ. Namun, jika daun berlabel Ɛ, maka harus sesuai rooth daun, ranting, cabang sampai dengan akarnya.

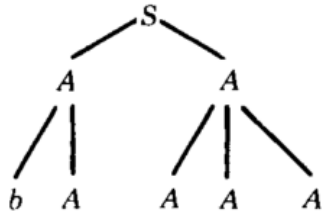
Kita mulai dengan simbol S. Setiap kali kita menggunakan produksi untuk menggantikan nonterminal atau variable V dengan string, dari garis ke bawah dari nonterminal untuk setiap karakter dalam string. Sebagai contoh CFG berikut,

$$S \rightarrow AA$$
$$A \rightarrow AAA \mid bA \mid Ab \mid a$$

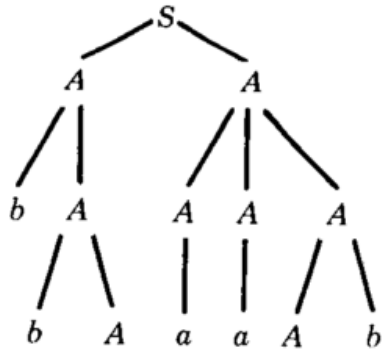
Kita mulai dari S → AA, dengan pohon urainya adalah



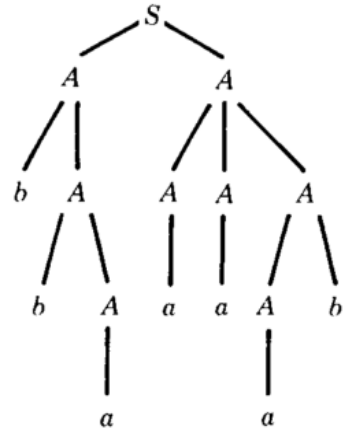
Kita mulai dari paling kiri A --> bA, kemudian kita menderevasi paling kanan A --> AAA, sehingga pohon urainya sebagaimana berikut



Kita selesaikan dari paling kiri dimana yang Non terminal kita derevasikan menjadi terminal, yaitu non terminal A → bA, A→ a, A→a, A→ Ab, sebagaimna gambar pohon urai dibawah ini

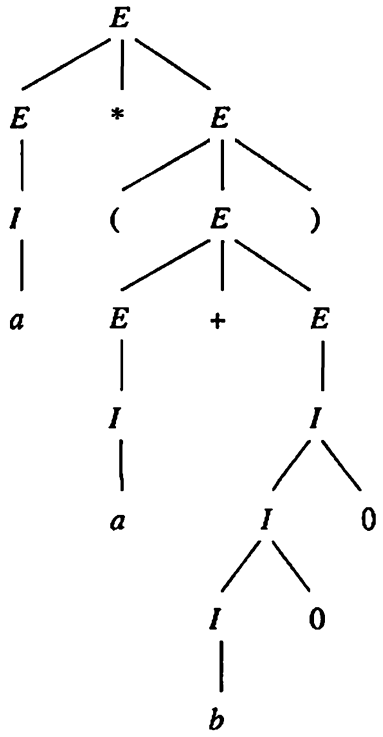


Kita harus selesaikan semua non terminal menjadi terminal yang akan menghasilkan w= bbaaaab.



Gambar 10.3: Gambar pohon urai CFG yang menghasilkan kata bbaaaab

Contoh CFG yang lain



Gambar 10.4: Pohon urai yang menghasilkan string atau kata a*{a+b00)

Bagaimana menggambarkan bagaimana tata bahasa yang menghasilkan string a*{a+b00). Artinya, tata bahasa G = (V, T, P, S), menghasilkan salah satu kata a*{a+b00)

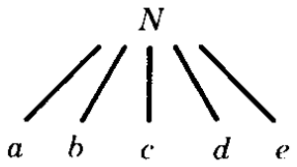
Membaca dari kiri ke kanan, kata yang dihasilkan adalah a*{a+b00) , seperti halnya dengan membuat diagram sebuah kalimat, kita lebih memahami tentang selesainya kata jika kita melihat seluruh pohon. Huruf dari paling kiri a, kedua * dan ketiga { yang dihasilkan oleh cabang-cabang pohon yang sama.

Diagram pohon ini disebut pohon sintaks atau pohon urai atau generasi pohon atau pohon produksi atau pohon turunan (derivasi) Berbagai nama datang dari banyaknya aplikasi hingga linguistik, desain kompiler, dan matematika logika.

Satu-satunya aturan untuk pembentukan pohon seperti itu adalah bahwa setiap tunas nonterminal cabang yang mengarah ke setiap karakter di sisi kanan produksi yang menggantikannya. Jika nonterminal N dapat diganti dengan string abcde:

$$N \rightarrow abcde$$

Gambar pohon urainya sebagai berikut

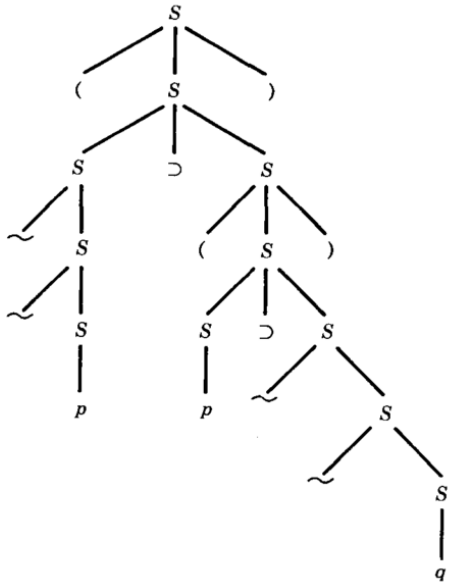


Satu CFG untuk subsistem Kalkulus adalah:

$$S \rightarrow (S) \mid S \supset S \mid \sim S \mid p \mid q$$

Dimana Non terminalnya adalah S, terminalnya adalah p q ~ ⊃ ()

Diagram pohon urainya



Hasil derivasinya adalah

$$(\sim\sim p \supset (p \supset \sim\sim q))$$

Definisi

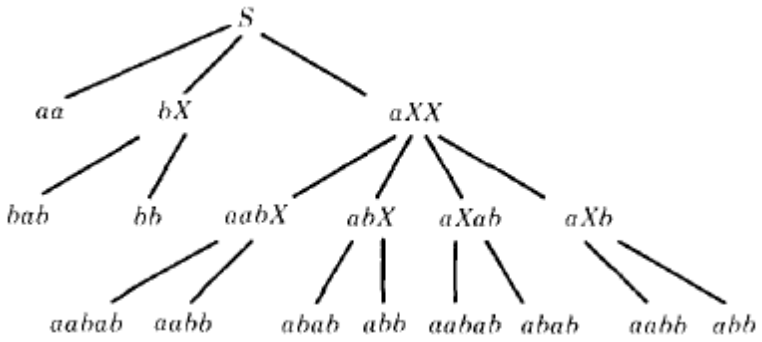
Untuk CFG yang diberikan, kami mendefinisikan pohon urai dengan simbol awal S sebagai akarnya dan node yang bekerja string terminal dan nonterminal. Derevasi dari setiap node adalah hasil dari semua yang mungkin dari penerapan setiap aturan produksi ke string, satu per satu. String dari semua terminal adalah terminal simpul di pohon. Pohon yang dihasilkan disebut pohon bahasa dari CFG.

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

CFG

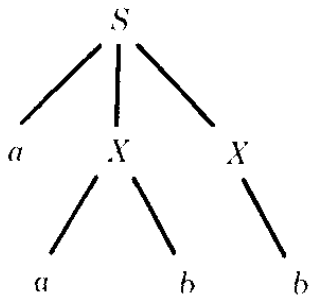
$$S \rightarrow aa \mid bX \mid aXX$$
$$X \rightarrow ab \mid b$$

Semua bahasa yang dinyatakan dalam pohon urai adalah



Gambar 10.5: Semua Bahasa yang Dinyatakan Dalam Pohon Urai

Semua Bahasa ini hanya memiliki tujuh kata yang berbeda. Empat dari kata-kata yaitu (abb, aabb, abab, aabab) memiliki dua kemungkinan derevasi yang berbeda karena mereka muncul sebagai simpul terminal dalam pohon urai semua bahasa ini di dua tempat berbeda. Namun, kata-kata tidak dihasilkan oleh dua pohon derivasi dan tata bahasa yang berbeda tidak ambigu. Sebagai contoh:



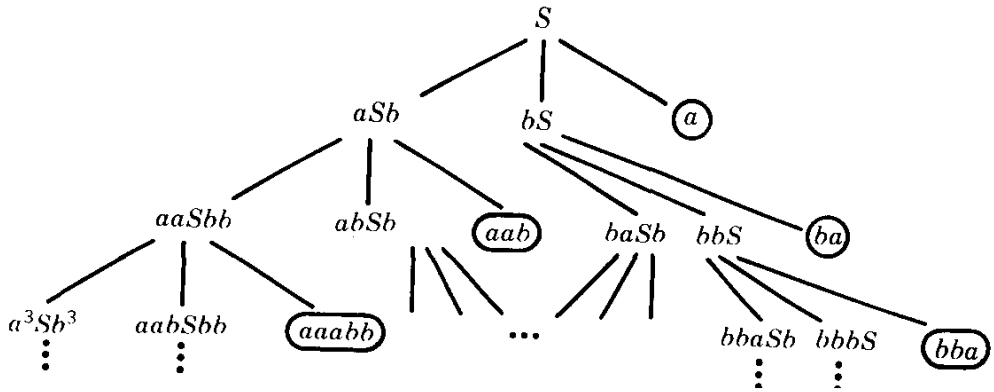
Contoh ;

CFG:

$$S \rightarrow aSb \mid bS \mid a$$

Kami memiliki huruf terminal a dan b dan tiga kemungkinan pilihan substitusi untuk S pada setiap tahap. Pohon total bahasa ini dimulai:

Kita sering mengatakan bahwa untuk mengetahui pohon derivasi untuk kata yang diberikan dalam bentuk tertentu tata bahasa adalah untuk memahami arti kata itu. Namun, di Kita akan segera menunjukkan dan mengetahui bahwa pohon itu membantu kita dapat membantu menentukan bagaimana mengevaluasi dan menghitung secara matematis dari hasil pohon derevasi.



Gambar 10.6: Melingkari Simpul Terminal

Di sini kita telah melingkari simpul terminal karena itu adalah kata-kata dalam bahasa yang dihasilkan oleh CFG ini. Kami mengatakan "dimulai" karena sejak bahasa tidak terbatas total pohon bahasa juga. Kami telah membuat semua kata dalam bahasa ini dengan satu, dua, atau tiga huruf

$$L = \{a,\ ba,\ aab,\ bba...\}$$

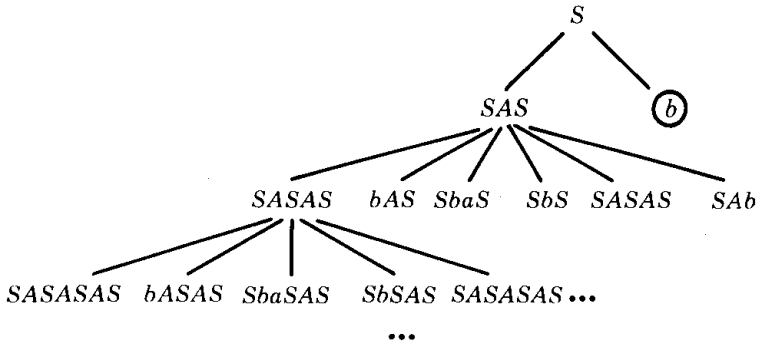
Pohon-pohon ini bisa menjadi lebar dan panjangnya tidak terbatas.

Contoh ;

S -- SAS | b

A -- ba | **b**

Setiap string dengan beberapa S dan beberapa A memiliki banyak kemungkinan produksi yang berlaku untuk itu, dua untuk setiap S dan dua untuk setiap A.

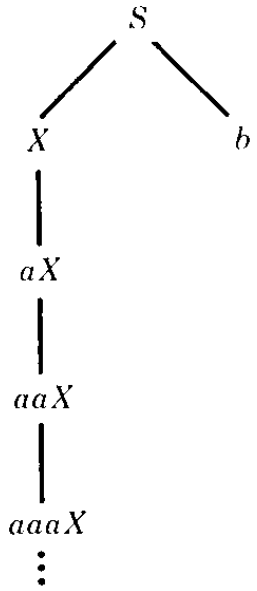


Contoh :

CFG:

$$S \rightarrow X \mid b$$
$$X \rightarrow aX$$

Semua bahasa yang dinyatakan dengan pohon urai sepeti gambar



Tatabahasa yang Ambiquitas

Contoh

CFG untuk versi ekspresi aritmatika:

$$S \rightarrow S + S \mid S * S \mid \underline{\text{number}}$$

yang dimaksud dengan "angka.", dan hasil yang ambiguitas dalam ekspresi sebagai berikut

3 + 4 * 5

Apakah artinya (3 + 4) * 5, yaitu 35, atau artinya 3 + (4 * 5), yang adalah 23?

Dalam bahasa yang ditentukan oleh tatabahasa CFG ini, tidak memiliki tanda kurung untuk klarifikasioperasi matematisnya. Tanda kurung tidak dihasilkan oleh salah satu produksi dan karena itu bukan huruf dalam bahasa turunan. Bahwa 3 + 4 * 5 adalah kata dalam bahasa CFG tidak memiliki aturan tetap sehingga terjadi perbedaan yang berarti yaitu apakah kata dari hasil derivasi 3 + 4 * 5 itu menghasilkan 35, atau kah 23

Kita lihat CFG berikut menggunakan tanda kurung di dalam CFGnya

$$S \rightarrow (S + S) \mid (S * S) \mid \underline{\text{number}}$$

Dari hasi derivasi CFG diatas maka menghasilkan

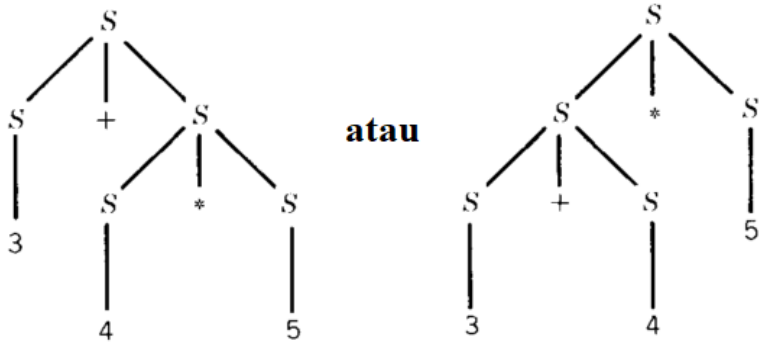
$$S \Rightarrow (S + S) \Rightarrow (S + (S * S)) \Rightarrow \ldots \Rightarrow (3 + (4 * 5))$$

Atau

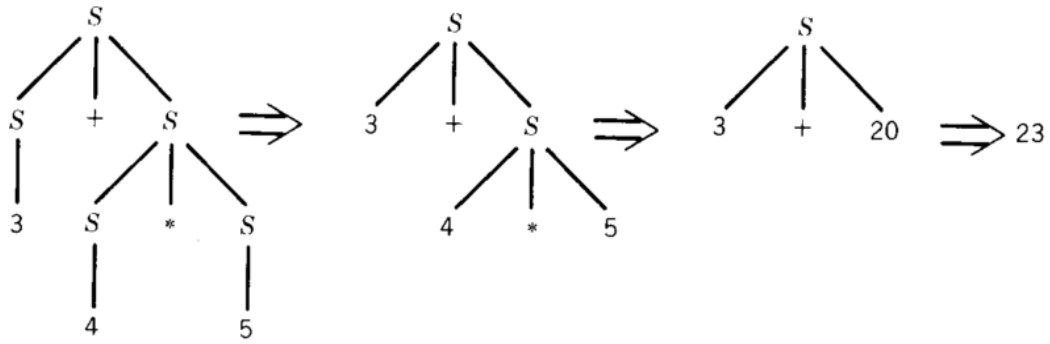
$$S \Rightarrow (S * S) \Rightarrow ((S + S) * S) \Rightarrow \ldots \Rightarrow ((3 + 4) * 5)$$

Dari ke dua ekspresi dan derivasi tidak ada hasil yang meragukan atau merupakan hasil yang tidak ambigu.

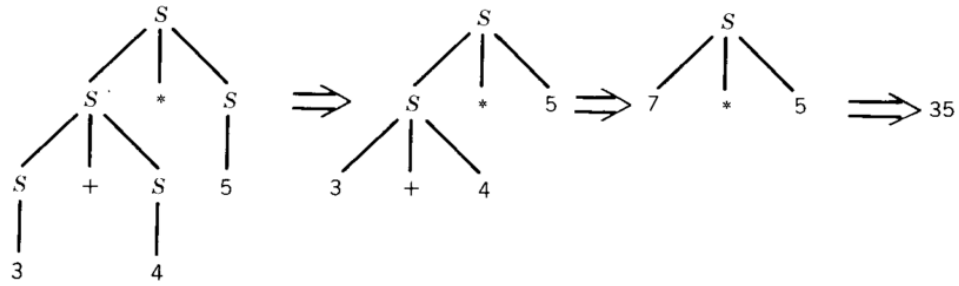
Kita lihatpohon urai yang menghasilkan kata yang ambiqity



Gambar 10.7: Diagram Kata yang Umbiguity



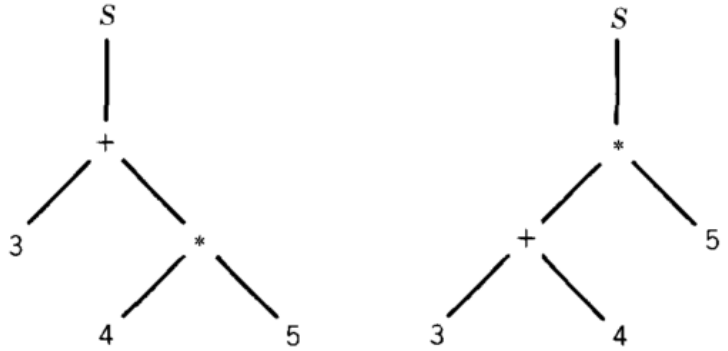
Gambar 10.8: Diagram kata yang menghasikan nilai 23



Gambar 10.9: Diagram kata yang menghasikan nilai 35

Contoh-contoh diatas menunjukkan bahwa pohon derivasi atau pohon turunan dapat menjelaskan kata apa, dan sebagaimana diartikan juga dicontohkan sama seperti yang dijelaskan oleh pohon turunan dalam bahasa Indonesia maupun bahasa inggis Inggris di dalam arti kalimat. Dalam kasus khusus tata bahasa ini (bukan untuk CFG secara umum), kita dapat menggambar pohon terminal yang bermakna menggunakan simbol awal S saja satu kali, ini menunjukkan untuk memperkenalkan notasi baru untuk ekspresi aritmatika yang memiliki aplikasi langsung dalam ilmu komputer

Metode menggambar pohon baru didasarkan pada fakta bahwa + dan * adalah operasi biner yang menggabungkan ekspresi yang sudah dalam bentuk yang tepat. Ekspresi 3 + (4 * 5) adalah jumlah. Jumlah dari apa? Jumlah suatu bilangan dan produk. Produk apa? Produk dari dua angka. Demikian pula (3 + 4) * 5 adalah produk dari jumlah dan angka, di mana jumlah adalah jumlah angka. Perhatikan kesamaan dengan definisi rekursif asli dari ekspresi aritmatika. Kedua situasi ini digambarkan dalam pohon berikut.



Ini seperti pohon turunan untuk CFG:

$$S \rightarrow S + S \mid S * S \mid \underline{\text{number}}$$

kecuali menghilangkan sebagian besar S. Simbol * dan + bukan lagi terminal, karena harus diganti dengan angka. Ini sebenarnya adalah pohon turunan standar yang diambil dari yang baru CFG di mana S, * dan + adalah nonterminal dan nomor adalah satu-satunya terminal.

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

Produksinya adalah:

$$S \rightarrow * \mid + \mid \underline{\text{number}}$$
$$+ \rightarrow ++ \mid +* \mid + \underline{\text{number}} \mid *+ \mid ** \mid * \underline{\text{number}} \mid \underline{\text{number}} + \mid \underline{\text{number}} * \mid \underline{\text{number}}\ \underline{\text{number}}$$
$$* \rightarrow ++ \mid +* \mid + \underline{\text{number}} \mid *+ \mid ** \mid * \underline{\text{number}} \mid \underline{\text{number}} + \mid \underline{\text{number}} * \mid \underline{\text{number}}\ \underline{\text{number}}$$

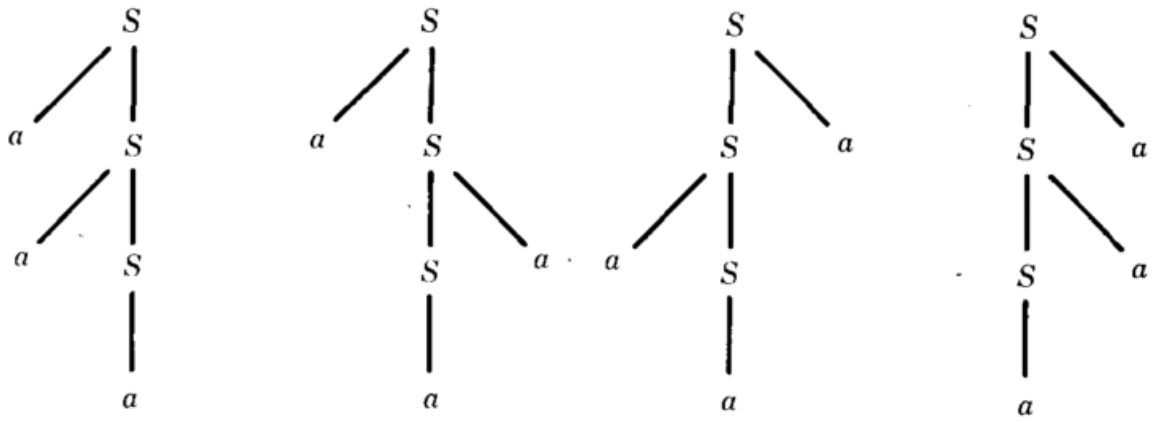
Contoh ini, kemudian, tidak menghadirkan kesulitan dalam substantifnya karena tidak ada interpretasi ysng ambiguitas. Ini terkait dengan situasi yang pertama kali membangun struktur tata bahasa dari kalimat bahasa Indonesia dan bahasa Inggris dari kata benda, kata kerja, dan sebagainya,

Definisi
Sebuah CFG disebut ambigu jika setidaknya satu kata dalam bahasa itu menghasilkan ada dua kemungkinan turunan dari kata yang memiliki pohon sintaks yang berbeda. Tata bahasa dapat menghasilkan lebih dari satu pohon urai untuk beberapa (atau semua) string yang dihasilkannya. Kapan ini terjadi kita mengatakan bahwa tata bahasanya ambigu. Lebih tepatnya, tata bahasa G ambigu jika setidaknya ada satu string di L(G) yang G menghasilkan lebih dari satu pohon urai.

Contoh
Bahasa yang terdiri dari semua string dan nullstring dari didefinisikan oleh CFG sebagai berikut:

$$S \rightarrow aS \mid Sa \mid a$$

Kita akan membuktikan kata aaa yang dihasilkan oleh CFG diatas dengan pohon urai



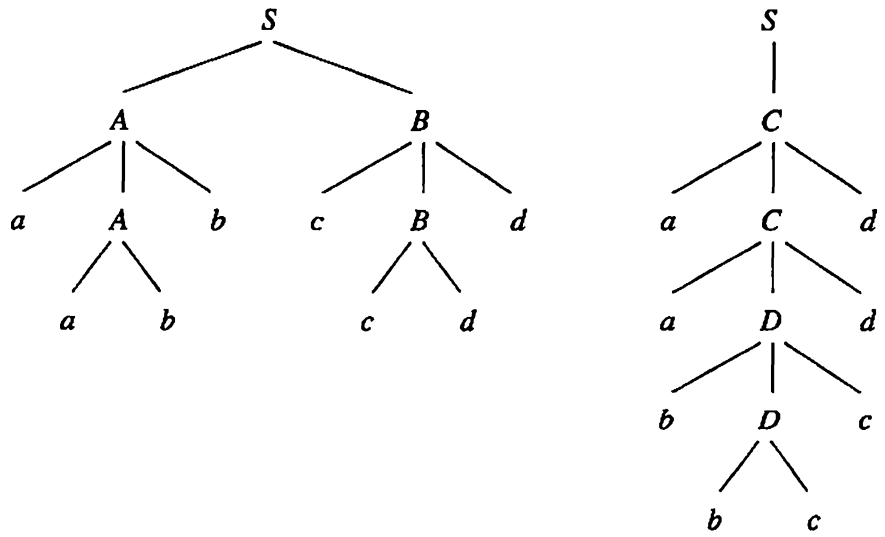
Gambar 10.10: Diagram Pohon Urai

Berdasarkan empat diagram pohon urai yang berbeda menunjukkan bahwa tatabahasa CFG diatas adalah ambiquity, karena mengkasilkan kata atau string aaa, didapatkan empat pohon yang berbeda.

Contoh CFG yang ambiquity

$$S \rightarrow AB \mid C$$
$$A \rightarrow aAb \mid ab$$
$$B \rightarrow cBd \mid cd$$
$$C \rightarrow aCd \mid aDd$$
$$D \rightarrow bDc \mid bc$$

CFG menghasilkan kata aabbccdd yang ambigu, dengan hasil pohon urainya berbeda



Gambar 10.11: Kata aabbccdd yang Ambigu, dengan Hasil Pohon Urainya Berbeda

Teknik untuk Mengurangi Ambiguitas

Terlepas dari hasil teoretis negatif yang baru saja kita sebutkan yaitu tatabahasa yang ambigu, biasanya sangat penting, ketika kita merancang bahasa kita harus menghasilkan bahasa yang tidak ambigu. Meskipun tidak ada algoritma untuk menguji ambiguitas dalam tata bahasa atau untuk menghapusnya ketika ditemukan (karena penghapusan tidak selalu mungkin), kami dapat menemukan beberapa sumber ambiguitas yang lebih umum dan menghapusnya. Tiga tata bahasa struktur yang sering menyebabkan ambiguitas:

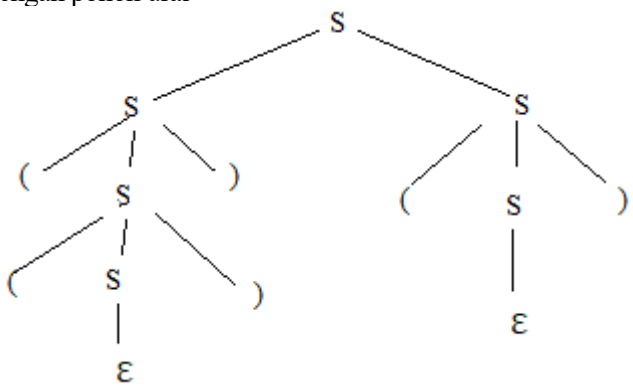
a) Aturan produksi seperti S → Ɛ
b) Aturan produksi seperti S → SS atau E → E + E. Dengan kata lain aturan rekursif yang ruas kanannya simetris dan berisi setidaknya dua salinan nonterminal di sisi kiri.
c) Kumpulan aturan produksi yang mengarah pada lampiran ambigu dari postfix opsional.

Menghilangkan aturan produksi Ɛ

Dalam Contoh CFG

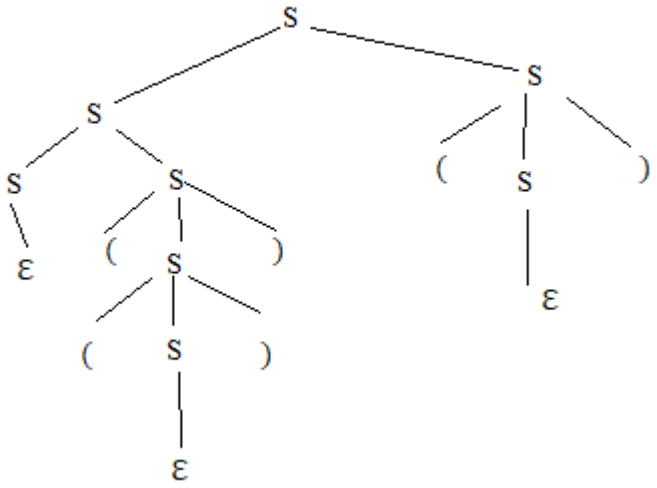
S → (S)
S → SS
S → Ɛ

Membuktikan string (())()dengan pohon urai



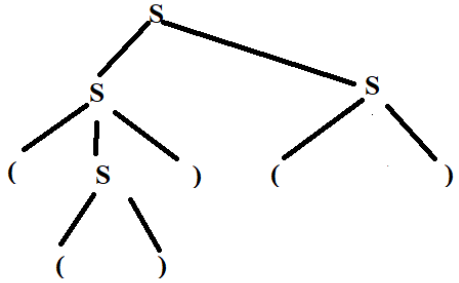
Gambar 10.12: Diagram pohon menghasilkan (())()

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress



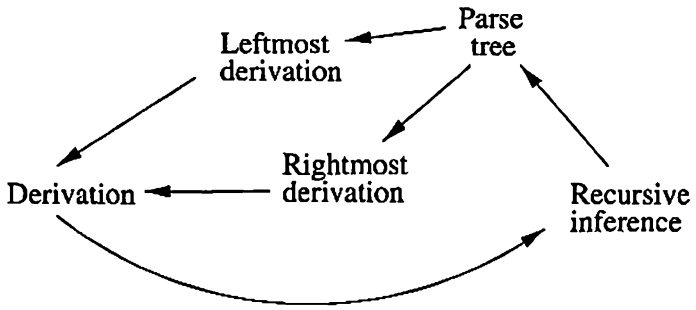
Gambar 10.12: Diagram pohon menghasilkan (())()

Contoh CFG
S → (S) | () | SS | S



Gambar 10.13: Contoh CFG

Leftmost derivation dan Right derivation
Prosedur inferensi rekursif menentukan bahwa string terminal w ada di bahasa variabel A.
1) A -> W, A dapat membangkitkan W
2) A -> W, A dapat membangkitkan W dengan leftmost derivation
3) A -> W, A dapat membangkitkan W dengan Rightmost derivation
4) Ada pohon parse dengan akar A dan hasil w.



Gambar 10.14: Dua Busur Sangat Sunple

Perhatikan bahwa dua busur sangat sunple dan tidak akan dibuktikan secara formal, jika w memiliki turunan paling kiri dari A, maka pasti memiliki turunan dari A karena turunan paling kiri adalah turunan. Demikian juga,

jika w memiliki turunan paling kanan, maka pasti ada turunannya. sekarang melanjutkan untuk membuktikan langkah-langkah yang lebih sulit dari kesetaraan ini.

Atau dengan kata alin bahwa leftmost derivation adalah di mana, pada setiap langkah, yang nonterminal paling kiri dipilih untuk di turunkan terlebih dahulu hingga mencapai terminal dalam string.

Rightmost derivation adalah di mana, pada setiap langkah, nonterminal yang paling kanan dulu yang diturunkan terlebih dahulu hingga mencapai terminal dalam string.

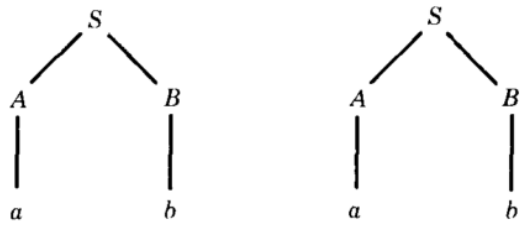
Mari kita perhatikan bahasa yang dihasilkan oleh CFG berikut:

$$\text{PROD 1} \quad S \rightarrow AB$$
$$\text{PROD 2} \quad A \rightarrow a$$
$$\text{PROD 3} \quad B \rightarrow b$$

Ada dua urutan derivasi yang berbeda dari aplikasi produksi yang menghasilkan kata ab. Salah satunya adalah PROD 1, PROD 2, PROD 3. Yang disebut turunan dari paling kiri sampai ke kanan atau disebut dengan leftmost derivation Yang lainnya adalah PROD 1,PROD 3, PROD 2. Diturunkan dari paling kanan dulu kemudian berurutan sampai deng paling kiri disebut dengan Rightmost derivation

$$S \Rightarrow AB \Rightarrow aB \Rightarrow ab \qquad \textbf{atau} \qquad S \Rightarrow AB \Rightarrow Ab \Rightarrow ab$$

Pohon urainya adalah

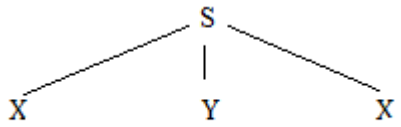


Gambar 10.15: Pohon Uraian

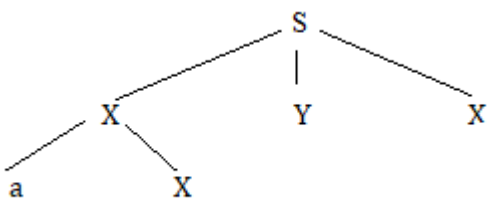
Contoh hasil dari leftmost derivation trees dan Rightmost derivation strees sebagai berikut

CFG
S→ XYX
X → aX | bX | Ɛ
Y → bbb

a. abbba dengan leftmost derivation trees

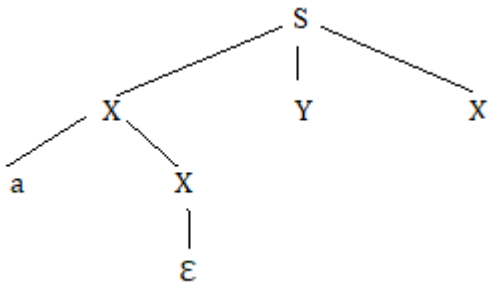


Gambar 10.16: Diagram S→ XYX

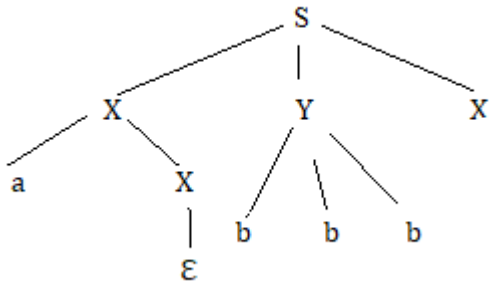


Gambar 10.17: Diagram S→ XYX → aXYX

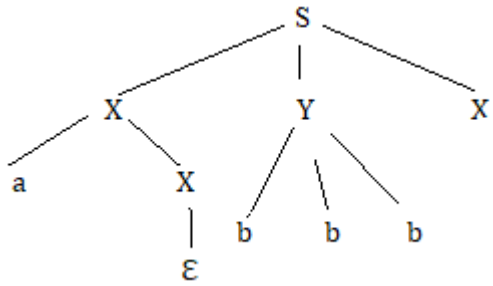
Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress



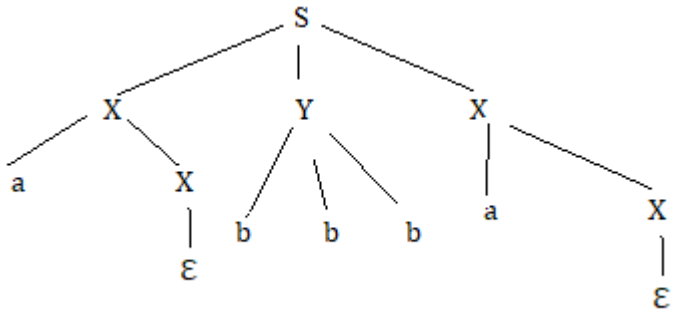
Gambar 10.18: Diagram S→ XYX → aXYX → aƐYX



Gambar 10.19: Diagram S→ XYX → aXYX → aƐYX → aƐbbbX

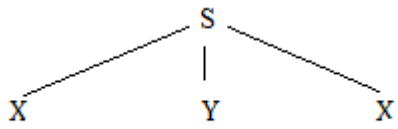


Gambar 10.20: Diagram S→ XYX → aXYX → aƐYX → aƐbbbX → aƐbbbaX

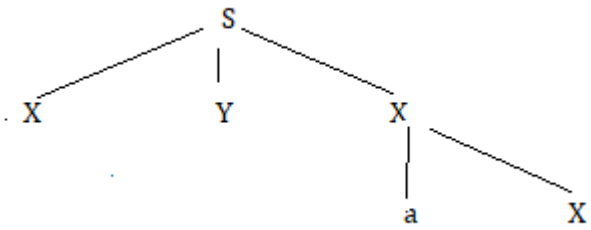


Gambar 10.21: Diagram S→ XYX → aXYX → aƐYX → aƐbbbX → aƐbbbaX → aƐbbbaƐ

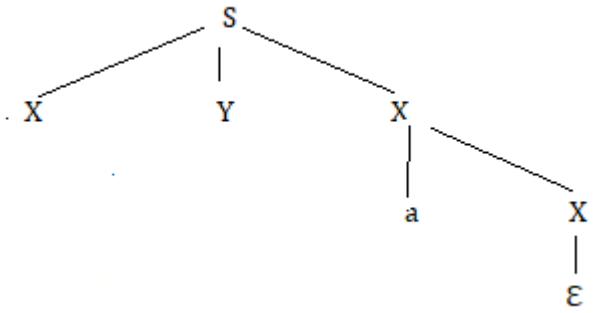
b. Rightmost Derivation trees



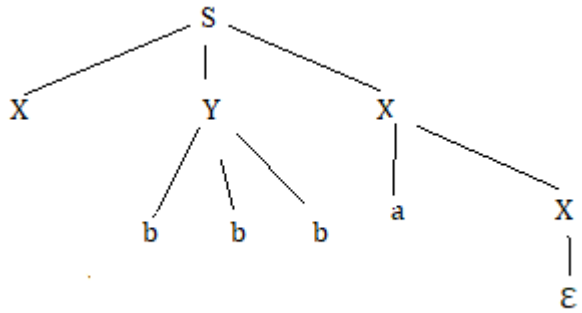
Gambar 10.22: Diagram S→ XYX → aXYX



Gambar 10.23: Diagram S→ XYX → XYaX

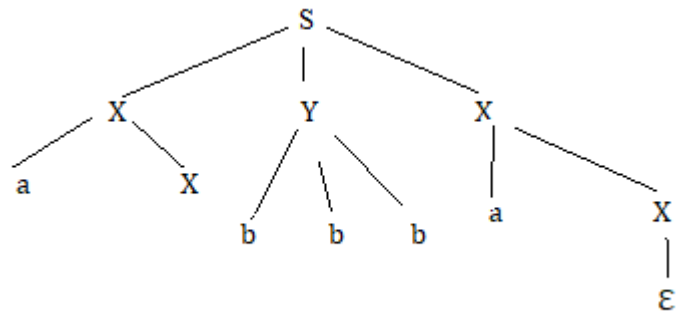


Gambar 10.24: Diagram S→ XYX → XYaX → XYaƐ

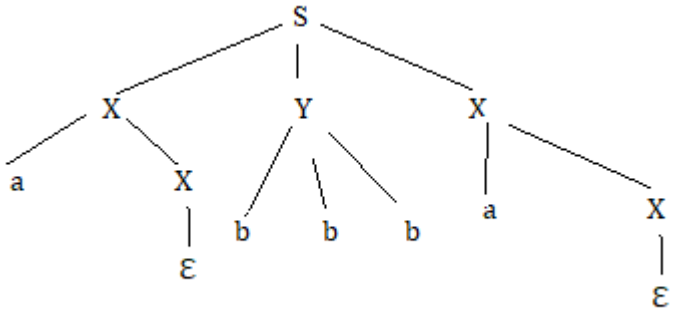


Gambar 10.25: Diagram S→ XYX → XYaX → XYaƐ → XbbbaƐ

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress



Gambar 10.26: Diagram S→ XYX → XYaX → XYaƐ → XbbbaƐ → aXbbbaƐ



Gambar 10.27: Diagram Rightmost derivation trees aƐbbbaƐ menjadi abbba

.

10.2 Latihan Soal

1. Di bawah ini adalah kumpulan kata dan kumpulan CFG. Untuk setiap kata, tentukan apakah kata tersebut dalam bahasa masing-masing CFG dan, jika ya, gambarkan pohon sintaks ke buktikan.

CFG's

CFG 1. $S \rightarrow aSb \mid ab$

CFG 2. $S \rightarrow aS \mid bS \mid a$

CFG 3. $S \rightarrow aS \mid aSb \mid X$
$X \rightarrow aXa \mid a$

String atau kata

ab

aaaa

aabb

abaa

abba

baaa

abab

bbaa

baab

2. Tunjukkan bahwa CFG berikut ambigu dengan mencari kata dengan dua pohon sintaks yang berbeda.

(i) $S \rightarrow SaSaS \mid b$

(ii) $S \rightarrow aSb \mid Sb \mid Sa \mid a$

(iii) $S \rightarrow aaS \mid aaaS \mid a$

(iv) $S \rightarrow aS \mid aSb \mid X$
$X \rightarrow Xa \mid a$

3. Tentukan No. 2 dengan Rightmost derivation dan Leftmost derivation trees.

Bab 11

Regular Grammar

11.1 Regular Grammar

Menurunkan bahasa dari Regular grammer

Beberapa contoh bahasa yang kami hasilkan oleh CFG adalah bahasa reguler, yaitu, mereka didefinisikan oleh ekspresi reguler. Namun, kami juga telah melihat beberapa bahasa nonreguler yang dapat dihasilkan oleh CFG (PALINDROME dan EQUAL).

Contoh :

CFG:

$$S \rightarrow ab \mid aSb$$

menghasilkan bahasa $\{a^n b^n\}$

Aplikasi berulang dari hasil produksi kedua dalam derivasi

S → aSb → aaSbb → aaaSbbb → aaaaSbbbb ...

Akhirnya produksi pertama akan diterapkan untuk membentuk kata yang memiliki kesamaan jumlah a dan b, dengan semua a pertama. Bahasa ini seperti yang kami tunjukkan pada bab sebelumnya, yang tidak regular.

Contoh :

CFG:

S → aSa | bSa | Ɛ

menghasilkan bahasa dari semua kata dalam bentuk: s alength(s) untuk semua string s di (a + b)* yaitu, string apa pun yang digabungkan dengan string sebanyak a yang dimiliki string. Bahasa ini juga nonregular, karena tidak dapat disefinisikan dengan regular ekpresi

Lalu apa hubungan antara bahasa reguler dan tata bahasa bebas konteks?

Beberapa kemungkinan muncul dalam pikiran:

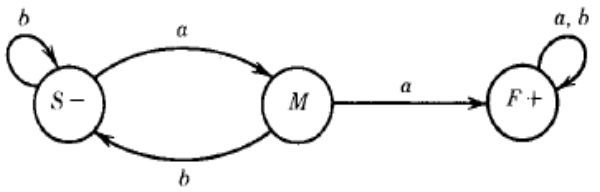
a. Semua bahasa dapat dihasilkan oleh CFG.
b. Semua bahasa reguler dapat dihasilkan oleh CFG, dan juga beberapa bahasa nonreguler tetapi mungkin tidak semua bahasa.
c. Beberapa bahasa reguler dapat dihasilkan oleh CFG dan beberapa bahasa nonreguler yang dihasilkan oleh CFG. Beberapa bahasa non-reguler dapat dihasilkan oleh CFG dan beberapa bahasa yang juga non-reguler.

Dari ketiga kemungkinan ini, nomor 2 yang benar. Dalam bab ini kita akan memang menunjukkan bahwa semua bahasa reguler dapat dihasilkan oleh CFG.

Kami sekarang menyajikan metode untuk mengubah FA menjadi CFG sehingga semua kata-kata yang diterima oleh FA dapat dihasilkan oleh CFG dan hanya kata-kata diterima oleh FA dihasilkan oleh CFG. Proses konversinya adalah lebih mudah dari yang kita duga. Hal ini, tentu saja, dinyatakan sebagai algoritma konstruktif yang pertama kita ilustrasikan pada contoh sederhana.

CONTOH

Mari kita pertimbangkan FA di bawah ini, yang menerima bahasa semua kata dengan a ganda:



Gambar 11.1: FA

Kami telah menamai state awal S,state tengah M, dan state akhir F. Kata abbaab diterima oleh mesin ini. Dari pada menelusuri mesin melihat bagaimana huruf inputnya dibaca, seperti biasa, mari kita lihat caranya

S (Kita mulai di S)

aM (Kami mengambil sisi-a ke M)

abS (Kami mengambil a-edge lalu b-edge dan kami berada di S)

abbS (a-edge, b-edge, dan b-loop kembali ke S)

abbaM (a-edge lain dan berada di M)

abbaaF (a-edge lain dan berada di F)

abbaabF (b-loop kembali ke F)

abbaab (Jalur yang sudah selesai: a-edge a b-edge . .)

Pengembangan jalur ini sangat mirip dengan turunan kata dalam CFG. maka menjadi aturan produksi?

(Dari S sebuah a-edge membawa ke M) S → aM

(Dari S b-sisi membawa kita ke S) S → bS

(Dari M sebuah a-edge membawa kita ke F) M → aF

(Dari M b-sisi membawa kita ke S) M → bS

(Dari F a-sisi membawa kita ke F) F → aF

(Dari F b-sisi membawa kita ke F) F → bF

(Ketika pada state akhir F, kita dapat F → Ɛ

berhenti jika kita mau)

Kami akan membuktikan sebentar lagi bahwa CFG yang baru saja kami jelaskan menghasilkan semua jalur dari S ke F dan karenanya menghasilkan semua kata yang diterima oleh FA. Mari kita perhatikan jalan lain dari S ke F, yaitu dari kata babbaaba. Itu urutan pengembangan jalur adalah

(Mulai di sini) S

(b-loop kembali ke S) bS

(a-sisi ke M) baM

(b-sisi kembali ke S) babS

(b-loop kembali ke S) babbS

(Sebuah sisi ke M) babbaM

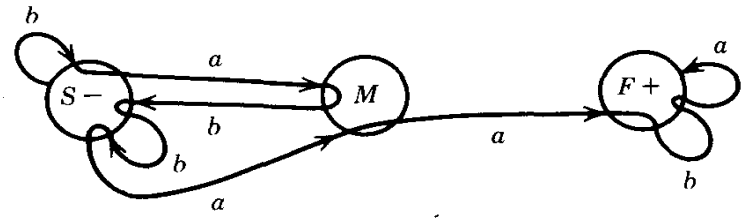
(Satu lagi sisi untuk F) babbaaF

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

(b-loop kembali ke F) babbaabF

(Sebuah putaran kembali ke F) babbaabaF

(Selesai di F) babbaaba



Gambar 11.4: Pengembangan Jalur

Ini bukan hanya pengembangan jalur tetapi juga turunan dari kata babbaaba dari CFG di atas. Logika argumen ini kira-kira sebagai berikut. Setiap kata diterima oleh FA ini sesuai dengan jalur dari S ke F. Setiap jalur memiliki langkah-demi-langkah urutan pengembangan seperti di atas. Setiap urutan pengembangan adalah turunan dalam CFG yang diusulkan. Oleh karena itu, setiap kata yang diterima oleh FA dapat dihasilkan oleh CFG.

Kebalikannya juga harus benar. Kita harus menunjukkan bahwa setiap kata yang dihasilkan oleh CFG ini adalah kata yang diterima oleh FA. Mari kita ambil beberapa turunan seperti sebagai

Penggunaan produksi produksi

S → aM

M →-bS

S → aM

M →-aF

F →- bF

F → Ɛ

Derivasi (turunan)

S → aM

→- abS

→ abaM

→ abaaF

→- abaabF

→ abaab

Ini dapat diartikan sebagai pengembangan jalur:

Penggunaan aturan produksi

S → aM

M → bS

S → aM

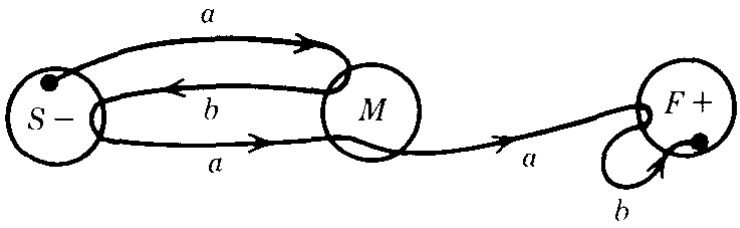
M → aF

F → bF

F → Ɛ

Pengembangan jalur

Mulai dari S kita ambil a-edge ke M
Kemudian b-sisi ke S
Kemudian a-sisi ke M
Kemudian a-sisi ke F
Kemudian b-sisi ke F
Sekarang kita berhenti



Gambar 11.5: Pengembangan Jalur

Definisi

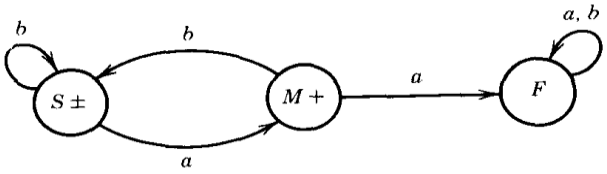
Untuk CFG yang diberikan, semiword adalah string terminal (mungkin tidak ada) yang digabungkan dengan tepat satu nonterminal (di sebelah kanan), misalnya,

(terminal) (terminal). . . (terminal) (Nonterminal)

Bandingkan ini dengan Word, yang merupakan string dari semua terminal, dan berfungsi string, yang merupakan string dari sejumlah terminal dan nonterminal di sembarang memesan.

Mari kita periksa selanjutnya kasus FA yang memiliki dua state akhir. Salah satu contoh mudahnya adalah FA untuk bahasa semua kata tanpa ganda a. Ini, pelengkap bahasa contoh terakhir, juga reguler dan diterima oleh mesin FA'.

FA'



Gambar 11.6: Kasus FA yang Memiliki Dua State Akhir

Mari kita simpan sejenak nama-nama nonterminal yang kita miliki sebelumnya:

S untuk state awal, M untuk tengah, dan F untuk apa yang dulunya merupakan state akhir, tetapi sekarang gambar diatas sudah tidak lagi. Produksi yang yang digambarkan sebagai berikut

$$S \rightarrow aM \mid bS$$
$$M \rightarrow bS \mid aF$$
$$F \rightarrow aF \mid bF$$

Namun, sekarang kami memiliki serangkaian state akhir yang berbeda. Kami dapat menerima string dengan jalurnya berakhiran S atau M, jadi produksinya:

$$S \rightarrow \varepsilon$$

dan

$$M \rightarrow \varepsilon$$

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

tapi tidak

F-→ Ɛ

Paragraf berikut adalah penjelasan mengapa algoritma ini bekerja: Jalur apa pun melalui mesin FA' yang dimulai pada - sesuai dengan string label tepi dan secara bersamaan ke urutan produksi yang menghasilkan a semiword yang bagian terminalnya adalah string label tepi dan ujung kanannya nonterminal adalah nama state dimana path berakhir. Jika path berakhir dengan a state akhir, maka kita dapat menerima string input sebagai kata dalam bahasa mesin, dan secara bersamaan menyelesaikan pembuatan kata ini dari CFG dengan menggunakan produksi:

(Nonterminal sesuai dengan state akhir) - Ɛ

Karena definisi tentang CFG mengharuskan selalu memulai derivasi dengan simbol awal S, selalu perlu untuk memberi label yang unik start state dalam FA dengan nama nonterminal S. Sisanya adalah pilihan nama negara adalah arbitrer.Diskusi ini bersifat umum dan cukup lengkap untuk dianggap sebagai bukti dari teorema berikut:

Teorema

Semua bahasa reguler dapat dihasilkan oleh CFG.

Ini juga dapat dinyatakan sebagai: Semua bahasa reguler adalah CFL.

Contoh

Bahasa semua kata dengan jumlah a genap (dengan setidaknya beberapa

a) adalah reguler karena dapat diterima oleh FA ini:

State S, M, dan F seperti sebelumnya, kita memiliki persamaan berikut:

set produksi:

S -→ bS AKU

M -→ bM I aF

F -→ bF | aM | Ɛ

Kami telah melihat dua CFG untuk bahasa ini, tetapi CFG ini secara substansial berbeda. (Di sini kita dapat mengajukan pertanyaan mendasar: Bagaimana kita bisa beri tahu apakah dua CFG menghasilkan bahasa yang sama? Tapi pertanyaan mendasar tidak selalu memiliki jawaban yang memuaskan.)

Teorema

Jika semua produksi dalam CFG dengan salah satu dari dua bentuk:

Nonterminal --> semiword

atau

Nonterminal -→ kata

(dimana kata mungkin Ɛ) maka bahasa yang dihasilkan oleh CFG ini adalah regular.

Bukti

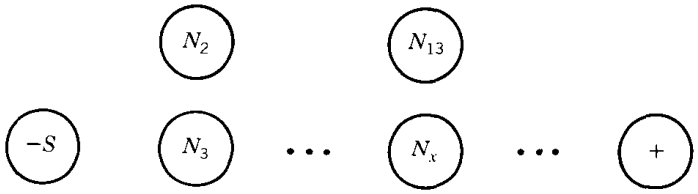
Kami akan membuktikan bahwa bahasa yang dihasilkan oleh CFG seperti itu beraturan dengan menunjukkan bahwa ada TG yang menerima bahasa yang sama. Kami akan membangun TG ini dengan algoritma konstruktif. Mari kita pertimbangkan CFG umum dalam bentuk ini:

$$N_1 \rightarrow w_1N_2 \qquad N_7 \rightarrow w_{10}$$
$$N_1 \rightarrow w_2N_3 \qquad N_{41} \rightarrow w_{23}$$
$$N_2 \rightarrow w_3N_4 \qquad \ldots$$

di mana N adalah nonterminal, w adalah string terminal, dan bagian wy Nz adalah semiword yang digunakan dalam produksi. Salah satu dari N ini harus

S. Misalkan N1 = S.

Gambarlah sebuah lingkaran kecil untuk setiap N dan satu lingkaran tambahan berlabel +. Lingkaran untuk S kita beri label -.

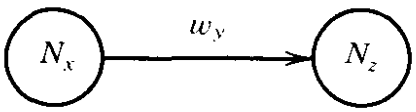


Gambar 11.7: Lingkaran N=S

Untuk setiap aturan produksi dalam bentuk:

Ux—wy Nz

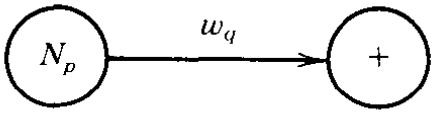
gambarlah tepi terarah dari keadaan Nx ke Nz dan beri label dengan kata wy.



Jika kedua nonterminal di atas sama, jalurnya adalah loop.Untuk setiap aturan produksi dalam bentuk:

Np – Wq

gambarlah tepi berarah dari Np ke + dan beri label dengan kata Wq.



Sekarang telah membangun transisi graph. Jalur apa pun di TG ini dari - ke + sesuai dengan kata dalam bahasa TG (dengan menggabungkan label) dan secara bersamaan sesuai dengan urutan produksi di CFG menghasilkan kata yang sama. Sebaliknya, setiap produksi kata di

CFG ini:

S → wN - → wwN - → wwwN ... = wwwww

sesuai dengan jalur di TG ini dari - ke +.

Oleh karena itu, bahasa TG ini sama persis dengan bahasa CFG. Oleh karena itu, bahasa CFG adalah reguler.

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

Kamu Kita harus mencatat bahwa fakta bahwa produksi di beberapa CFG semuanya masuk format yang diperlukan tidak menjamin bahwa tata bahasa menghasilkan kata apa pun. Jika tata bahasanya benar-benar terpecah, TG yang kita bentuk darinya akan menjadi gila juga dan tidak menerima kata-kata. Namun, jika tata bahasa menghasilkan bahasa dari beberapa kata maka TG yang dihasilkan di atas akan menerimanya bahasa.

Definisi

Sebuah CFG disebut tata bahasa reguler jika setiap produksinya adalah salah satu dari dua bentuk

Nonterminal -→ semiword (semikata)

atau

Nonterminal -→ word (kata)

Dua bukti sebelumnya menyiratkan bahwa semua bahasa reguler dapat dihasilkan oleh tata bahasa reguler dan semua tata bahasa reguler menghasilkan bahasa reguler.

Kita harus sangat berhati-hati untuk tidak terbawa oleh simetri ini teorema. Terlepas dari kedua teorema, masih mungkin bahwa CFG yang tidak masuk bentuk tata bahasa yang teratur dapat menghasilkan bahasa regular. Sebenarnya kita telah melihat contoh dari fenomena ini di Bab Bab sebelumnya.

Contoh

Pertimbangkan CFG:

S -→ aaS | bbS | Ɛ

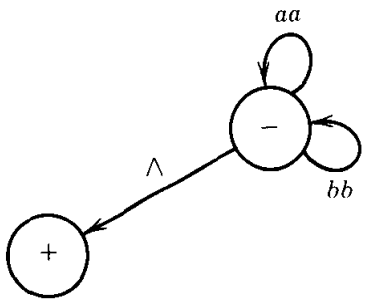
Ini adalah tata bahasa biasa dan jadi kami dapat menerapkan algoritme untuk itu. Di sana hanya satu nonterminal, S, jadi hanya akan ada dua keadaan di TG, - dan mandat +. Satu-satunya bentuk produksinya

Np -→ Wq

adalah

S -→ Ɛ

jadi hanya ada satu sisi ke + dan itu diberi label Ɛ. Produksi S -→ aaS dan S --> bbS berbentuk N1 -→ wN2 dimana N keduanya adalah S. Karena ini seharusnya dibuat menjadi jalur dari N1 ke N2 mereka menjadi loop dari S kembali ke S. Kedua produksi ini akan menjadi dua loop di - satu berlabel aa dan satu berlabel bb. Seluruh TG ditunjukkan di bawah ini:



Gambar 11.: TG

Dengan teorema Kleene, bahasa apa pun yang diterima oleh TG adalah reguler, oleh karena itu bahasa yang dihasilkan oleh CFG ini (yang sama) adalah reguler. Itu sesuai dengan ekspresi reguler

(aa + bb)*

Contoh

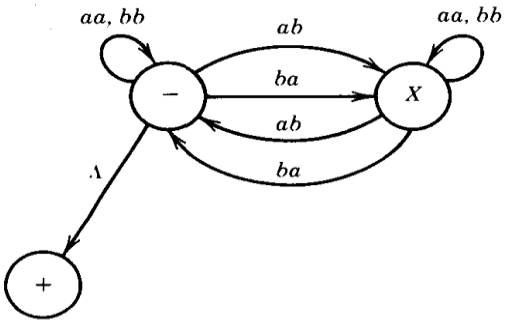
Pertimbangkan CFG:

S-- aaS l | bbS | abX | baX | Ɛ
X-- aaX | bbX | abs | baS

Algoritmanya ada tiga status: -, X, +. Karena hanya ada satu bentuk produksi formulir

Np --> Wq

hanya ada satu sisi ke +. TGnya adalah:

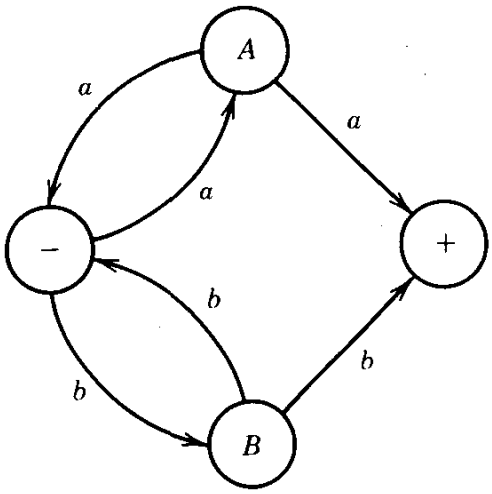


Contoh

Pertimbangkan CFG:

S --> aA | bB
A --> aS | a
B -→ bS | b

TGnya adalah:



Bahasa CFG ini persis sama dengan bahasa CFG dua contoh yang lalu kecuali bahwa itu tidak termasuk kata A. Bahasa ini dapat didefinisikan dengan ekspresi reguler (aa + bb)+.

11.2 Kekokohan Laba

Tentukan CFG yang menghasilkan bahasa reguler ini di atas alfabet

S= {a, b}:

1. Bahasa yang didefinisikan oleh (aaa + b)*
2. Bahasa yang didefinisikan oleh (a + b)* (bbb + aaa) (a + b)*
3. Semua string tanpa substring aaa.
4. Semua string yang diakhiri dengan b dan memiliki jumlah b yang genap.
5. Himpunan semua string dengan panjang ganjil.
6. Semua string dengan tepat satu a atau tepat satu b.
7. Semua string dengan bilangan ganjil a atau bilangan genap b.

Untuk CFG berikut temukan ekspresi reguler yang mendefinisikan bahasa yang sama dan mendeskripsikan bahasa.

8. S →aX │bS│ a│b

   X → aX│ a

9. S-→ bS │aX │b

   X → bX │as │a

10. S-→aaS│ abS│baS│bbS│Ɛ

11. S-→aB │bA │ Ɛ

    A -→ sebagai S

    B -→ bS

12. S-→aB│ bA

    A --> aB a

    B --> bA│ b

13. S-→aS │bX │a

    X-→ aX │oleh a

    Y--> aY│ a

14. S--> aS bXl a

    X-→ aXI oleh bZ a

    Y--> aY│ a

    Z--> aZ bW

    W --> aW a

15. S → bS│ aX

    X--> bS I│ aY

    Y-→ aY│ bY│a│b.

Bab 12

Chomsky Normal Form (CNF)

12.1 Chomsky Normal Form (CNF)

Tata bahasa bebas konteks datang dalam berbagai bentuk.sementara menurut definisi, setiap string terbatas terminal dan nonterminal adalah sisi kanan hukum dari produksi, misalnya

$$X \rightarrow YaaYbaYXZabYb$$

Berbagai kemungkinan memberi kebebasan pada kita yang cukup besar, tetapi juga menambah kesulitan pada kita menganalisis bahasa. Kita telah melihat di bab sebelumnya bahwa mungkin penting untuk mengetahui bentuknya dari tata bahasa. Dalam bab ini, kita akan menunjukkan bahwa semua bahasa bebas konteks dapat didefinisikan oleh CFG. Masalah pertama yang diatasi adalah null String atau himpunan kosong di dalam tatabahasa CFG. Itu permasalahan didalam FA dan TG,aturan produksi masalah seperti bentuk,

N → null tring

di mana N adalah sembarang nonterminal.Produksi →nullstring ini akan membuat permasalan yang sulit yang akan datang, karena Setiap bahasa bebas konteks di mana nullstring adalah kata harus memiliki beberapa produksi nullstring, pengurangan nullstring dalam tata bahasaharus dilakukan karena jika tidak, kita tidak akan pernah bisa menurunkan kata nullstring dari S. Pernyataan ini jelas, tetapi harus diberikan beberapa pembenaran.

Produsi-produksi nullstring adalah satu-satunya produksi yang memperpendek string kerja. Jika kita mulai dengan string S dan hanya menerapkan produksi non nullstring, kita tidak pernah mengembangkan kata yang panjangnya 0.

Namun, ada beberapa tata bahasa yang menghasilkan bahasa yang tidak termasuk kata nullstring tetapi tetap mengandung beberapa produksi nullstring. Salah satunya CFG yang sudah kita temui adalah

$$S \rightarrow aX$$
$$X \rightarrow e$$

Teorema karya Bar-Hillel, Perles, dan Shamir, menunjukkan bahwa produksi e tidak diperlukan dalam tata bahasa, untuk bahasa bebas konteks yang tidak mengandung kata e membuktikan hasil yang lebih baik

Teorema 21

Jika L adalah bahasa bebas konteks yang dihasilkan oleh CFG yang mencakup produksi e,lalu ada tata bahasa bebas konteks yang berbeda yang tidak memiliki produksi e yang menghasilkan baik seluruh bahasa L (jika L tidak termasuk kata e) atau yang lain menghasilkan bahasa semua kata dalam L yang bukan e.

Bukti

Membuktikan bahwa dengan menyediakan algoritma konstruktif yang akan mengubah CFG yang mengandung produksi e menjadi CFG yang tidak mengandung produksi e yang menghasilkan bahasa yang sama. Perhatikan CFG berikut untuk EVENPALINDROME (bahasa semua palindrom dengan jumlah huruf genap):

$$S \rightarrow aSa \mid bSb \mid e$$

Maka derivasinya sebagai berikut;

$$S \Rightarrow aSa$$
$$\Rightarrow aaSaa$$
$$\Rightarrow aabSbaa$$
$$\Rightarrow aabbaa$$

Mari kita lihat apa yang terjadi ketika kita menerapkan aturan penggantian ini ke menurunkan CFG.

$$S \rightarrow aSa \mid bSb \mid e$$

Kita menghapus produksi S □ A dan menggantinya dengan S □ aa dan S □ bb, yang merupakan dua produksi pertama dengan sisi kanan S dihapus. Maka CFG menjadi;

$$S \rightarrow aSa \mid bSb \mid aa \mid bb$$

Aturan Penggantian yang Dimodifikasi

a. Hapus semua produksi nullstring.
b. Tambahkan produksi berikut: Untuk setiap produksi

X → string lama

tambahkan cukup produksi baru dari bentuk X → bahwa sisi kanan akan memperhitungkan modifikasi string lama yang dapat dibentuk dengan menghapus semua kemungkinan subset dari nonterminal yang dapat dibatalkan, kecuali bahwa tidak ada X -> e akan dibentuk bahkan jika semua karakter dalam string sisi kanan lama ini dapat dibatalkan.

Contoh,

$$S \rightarrow a \mid Xb \mid aYa$$
$$X \rightarrow Y \mid \Lambda$$
$$Y \rightarrow b \mid X$$

bahwa X dan Y adalah nullable. Jadi ketika menghapus X → A kita harus periksa semua produksi yang menyertakan X atau Y untuk melihat produksi baru apa yang akan ditambahkan:

Nullables	**Produksi Baru**
X → Y	*Tidak ada*
X → e	*Tidak ada*
Y → X	*Tidak ada*
S → Xb	*S → b*
S → aYa	*S → aa*

Sehingga CFG barunya adalah;

$$S \rightarrow a \mid Xb \mid aYa \mid b \mid aa$$
$$X \rightarrow Y$$
$$Y \rightarrow b \mid X$$

CFG yang baru diatas tidak memiliki produksi nullstring tetapi menghasilkan bahasa yang sama dengan CFg dengan produksi nullstring.

Bagaimana jika tata bahasanya rumit?

Misalnya CFG

$$S \rightarrow Xay \mid YY \mid aX \mid ZYX$$
$$X \rightarrow Za \mid bZ \mid ZZ \mid Yb$$
$$Y \rightarrow Ya \mid XY \mid e$$
$$Z \rightarrow aX \mid YYY$$

semua nonterminal dapat dihapus, seperti yang dapat kita lihat dari

S→ZYX □YYYYX→YYYYZZ→YYYYYYYZ→YYYYYYYYYY→…….ɛɛɛɛɛɛɛɛɛ

Karena itu maka perlu dipertimbangkan untuk penghapusan dan penggantian baru kalau terdapat CFG yang memiliki produksi nullstring atau epsilon

Contoh

Mari kita perhatikan CFG berikut untuk bahasa yang didefinisikan oleh (a + b)*a

$$S \rightarrow Xa$$
$$X \rightarrow aX \mid bX \mid \varepsilon$$

Tabel 12.1: CFG

Nullables	**Produksi Baru**
S →Xa	*S→a*
X→ aX	*X→a*
X→bX	*X→b*

Sehingga CFG barunya;

$$S \rightarrow Xa \mid a$$
$$X \rightarrow aX \mid bX \mid a \mid b$$

Pertimbangkan CFG yang tidak efisien ini untuk bahasa yang didefinisikan oleh,

(a + b)*bb(a + b)*

$$S \rightarrow XY$$
$$X \rightarrow Zb$$
$$Y \rightarrow bW$$
$$Z \rightarrow AB$$
$$W \rightarrow Z$$
$$A \rightarrow aA \mid bA \mid \mathcal{E}$$
$$B \rightarrow Ba \mid Bb \mid \mathcal{E}$$

Dari X dapat menurunkan kata yang berakhiran b; dari Y dapat menurunkan apa saja kata yang dimulai dengan b. Oleh karena itu, dari S dapat menurunkan kata apa pun dengan dobel b. Jelas, A dan B adalah nullable. Berdasarkan itu, Z □ AB membuat Z juga tidak dapat dihapus. Setelah itu, kita melihat bahwa W juga dapat dihapus. X, Y, dan S tetap. Algoritma yang dimodifikasisecara bergantian untuk menghasilkan produksi baru untuk menghilangkan produksi nullable sebagai berikut:

CFG yang lama,

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

$$X \rightarrow Zb$$
$$Y \rightarrow bW$$
$$Z \rightarrow AB$$
$$W \rightarrow Z$$
$$A \rightarrow aA$$
$$A \rightarrow bA$$
$$B \rightarrow Ba$$
$$B \rightarrow Bb$$

CFG yang baru dari hasil derevasi yang lama menjadi,

$$X \rightarrow b$$
$$Y \rightarrow b$$
$$Z \rightarrow A \text{ and } Z \rightarrow B$$
Nothing
$$A \rightarrow a$$
$$A \rightarrow b$$
$$B \rightarrow a$$
$$B \rightarrow b$$

tidak menghilangkan semua produksi lama, hanya menghilangkan produksi nullable dan menambahkan dengan unit atau produksi baru, sehingga CFG baru yang tanpa nullable sepenuhnya dimodifikasi adalah:

$$S \rightarrow XY$$
$$X \rightarrow Zb \mid b$$
$$Y \rightarrow bW \mid b$$
$$Z \rightarrow AB \mid A \mid B$$
$$W \rightarrow Z$$
$$A \rightarrow aA \mid bA \mid a \mid b$$
$$B \rightarrow Ba \mid Bb \mid a \mid b$$

Teorema

Jika L adalah bahasa yang dihasilkan oleh beberapa CFG, maka ada CFG lain yang menghasilkan semua kata non Ɛ dari L, yang semua produksinya adalah salah satu dari dua bentuk dasar:

Nonterminal.--> string hanya Nonterminal

Atau

Nonterminal → satu terminal

Bukti

Buktinya dengan algoritma yang konstruktif. Misalkan dalam CFG yang diberikan nonterminalnya adalah S, X1, X2.....

(Jika ini sebenarnya bukan nonterminal di CFG, kita dapat mengganti nama mereka tanpa mengubah bahasa akhir. Biarkan Y menjadi X1 , misalkan N disebut X2 ....) diasumsikan juga bahwa terminalnya adalah a dan b. Kita sekarang menambahkan dua nonterminal baru A dan B dan produksi-produksinya adalah,

A → a

B → b

Sekarang untuk setiap produksi sebelumnya yang melibatkan terminal, kita mengganti masing-masing a dengan nonterminal A dan masing-masing b dengan nonterminal B. Misalnya,

$$X_3 \rightarrow X_4aX_1SbbX_7a$$

Menjadi

$$X_3 \rightarrow X_4AX_1SBBX_7A$$

Contoh non terminal memproduksi rangkaian semua terminal

$$X_6 \rightarrow aaba$$

Dikonversikan ke produksi semua Non terminal menjadi,

$$X_6 \rightarrow AABA$$

Semua produksi lama kita bentuk,

$$\text{Nonterminal} \rightarrow \text{string of Nonterminals}$$

dan dua produksi baru dalam bentuk

$$\text{Nonterminal} \rightarrow \text{one terminal}$$

Derivasi sebelumnya dimulai dari S dan berlanjut ke kata aaabba sekarang akan mengikuti urutan produksi yang sama untuk mendapatkan string

AAABBA

dari simbol awal S. Dari sini kita terapkan A → a dan B → b sebuah bilangan untuk menghasilkan kata aaabba, meyakinkan bahwa kata apa pun itu dapat dihasilkan oleh CFGjuga dapat dihasilkan oleh yang baru

Bentuk normal Chomsky (CNF) adalah salah satu bentuk normal yang berguna untuk Context Free Grammar (CFG) . CNF dapat dibuat dengan mengubah CFG dan penyederhanakan yaitu bagaimana penghilangan aturan produksi useless, unit, dan nulltring. Dengandemikian bahwa, suatu tata bahasa CFG dapat dikonversi menjadi CNF dengan ketentuan bahwa tababahasa tersebut harus :

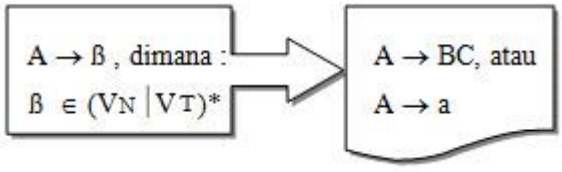
- ✓ Tidak terdapat atau tidak memiliki aturan produksi /useless
- ✓ Produksi unit
- ✓ Produksi ε

Bentuk normal Chomsky(CNF) adalah Context Free Grammar (CFG) dengan setiap produksi-produksi nya berbentuk :

A → XY atau A → a.

Perubahan dari Context Free Grammar ke Bentuk normal Chomsky

Perubahan dari bentuk tatabahasa CFG ke tatabahasa CNF adalah sebagai berikut :



Gambar 12.1: Perubahan dari Context Free Grammar ke Bentuk normal Chomsky

Perubahan dari CFG CFG ke CNF

Aturan produksi-produksinya dalam bentuk normal Chomsky pada sisi ruas kanan tepat berupa dua buah non terminal (dua huruf besa) dan sebuah terminal

Misalkan:

- ✓ A → XY (dua variable huruf besar)

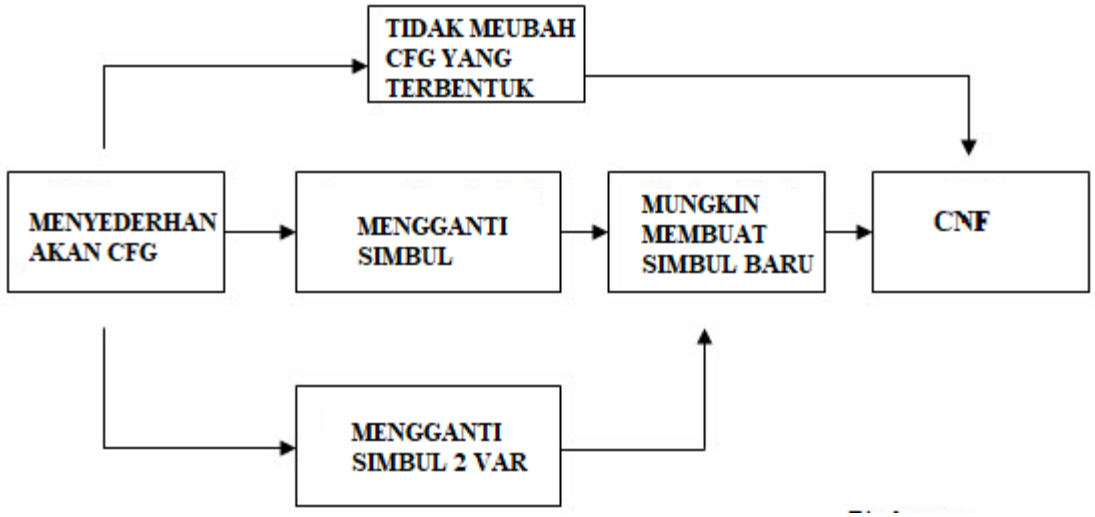
Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

- ✓ X → a (satu terminal/satu huruf kecil)
- ✓ Y → b(satu terminal/satu huruf kecil)
- ✓ C → DA | c

Pembentukan CNF

a. Tahapan pembentukan sebagai berikut:
b. Tidak merubah atau membiarkan aturan produksi yang sudah mengandung CNF
c. Melakukan penggantian aturan produksinya sisi ruas kanannya > 2 simbol variable
d. Kemungkinan membuat simbul baru
e. Mengganti secara berulang hinggga terbentuk ke bentuk normal Chomsky

Bisa dilihat tahapan-tahapan tersebut pada gambar berikut:



Gambar 12. 2: Langkah dan tahapan dari CFG ke CNF

1. Contoh, Context Free Grammar (kita anggap CFG pada bab ini sudah mengalami Penyederhanaan CFG):

   S → bA | aB

   A → bAA | aS | a

   B → aBB | bS | b

Aturan produksi yang sudah mmemenuhi ke bentuk normal Chomsky:

A → a

B → b

Tahapan yang dilakukan sebagai berikut :

S → bA => S → H1A

S → aB => S → H2B

A → bAA =>A → H1AA => A → H1H3

A → aS => A → H2S

B → aBB => B → H2BB => B → H2H4

B → bS => B → H1S

Terbentuk variabel baru:

H1 → b

H2 → a

H3 → AA

H4 → BB

Hasil bentuk normal Chomsky :

A → a

B → b

S → H1A

S → H2B

A → H1H3

A → H2S

B → H2

B → H1S

H1 → b

H2 → a

H3 → AA

H4 → BB

CNFnya menjadi:

S → H1 A | H2B

A → H1H3 | H2S | a

B → H2H4 | H1S | a

H2 → b

H1 → a

H3 → AA

H4 → BB

2. Contoh CFG:

S →aB | CA

A →a | bc

B →BC | Ab

C →aB | b

Aturan produksi yang sudah mmemenuhi ke bentuk normal Chomsky S → CA

A → a

B → BC

C → b

Penggantian aturan produksi:

S → aB => S → AB

A → bc => A → BC

B → Ab => B → AB

C → aB => C → AB

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

Terbentuk variabel baru:

A → a

B → b

C → c

Hasil akhir

S → AB | CA

A → a | BC

B → BC | AB

C→ AB | b

A→ a

B→ b

C→ c

3. Contoh dari CFG ke CNF lainya:

CFG:

S → aAB | ch | CD

A → dbE | eEC

B → ff | DD

C → ADB | aS

D → i

E → jD

Aturan produksi yang sudah terbentuk

S → CD

B → DD

D → i

Proses penggantian aturan,

S → aAB => S → PQ

S → ch => S → XY

A → dbE => A → MNE=> A → MO

A → eEC => A → FG

B → ff => B → HH

C → ADB => C → AI

C → aS => C → PS

E → jD => E → DD

Terbentuk aturan produksi yang baru:

P → a

Q→ AB

X→ c

Y→ h

M→d

N→b

O→NE

F →e

G →EC

H→ f

I → DB

L→ i

bentuk normal Chomskynya:

S→ PQ | SX | CD

A → MO | FG

B → HH | DD

C→ AI | PS

D→ i

E → DD

P → a

Q→ AB

X→ c

Y→ h

M→d

N→b

O→NE

F →e

G →EC

H→ f

I → DB

Jika CFG terdapat aturan produksi Nullstring, maka terlebih dahulu kita harus hilangkan

1. Contoh penyederhanaan CFG:

   A→ AB | ab

   B → BaC| C

   C → bC| ε

Menghilangkan Produksi-produksi yang mengandung nullable (ε)

Menghilangkan semua hasil produksi yang ε, dengan ;

Jika X→ ε, maka X adalah nullstring , Jika Y → X, maka Y adalah nullstring, Jika Z→ Xa, maka Z→ a

Demikian hingga CFG di atas, menjadi CFG tidakmemiliki produksi ε

$$A \rightarrow cAB \mid ab \mid cA$$

$B \rightarrow BaC \mid C \mid Ba \mid aC \mid a$

$C \rightarrow bC \mid b$

2. Contoh paling sederhana CFG

   S→ aS | ε, menjadi S→ aS | a

3. Contoh CFG

   S→ bcAd | ε

   A→bd

   Maka setelah menghilangkan produksi nullablenya CFG menjadi

   S→ bcAd | bcd

   A→bd

4. Contoh CFG;

   S→ AA → C → bd

   A→ Bb → Ɛ

   B→ AB → d

   C→ de

   Dengan menghilangkan produksi yang mengandung nullable mka CFg menjadi

   S→ AA → C → bd → A

   A→ Bb

   B→ AB → d → B

   C→ de

5. Ditunjukkan CFG yang yang didefinisikan dengan ekpresi regular, Sederhanan dengan CFG tanpa nullable

$$(\mathbf{a} + \mathbf{b})^*\mathbf{bb}(\mathbf{a} + \mathbf{b})^*$$

$S \rightarrow XY$
$X \rightarrow Zb$
$Y \rightarrow bW$
$Z \rightarrow AB$
$W \rightarrow Z$
$A \rightarrow aA \mid bA \mid \varepsilon$
$B \rightarrow Ba \mid Bb \mid \varepsilon$

CFG AWAL	CFG BARU
$X \rightarrow Zb$	$X \rightarrow b$
$Y \rightarrow bW$	$Y \rightarrow b$
$Z \rightarrow AB$	$Z \rightarrow A$ and $Z \rightarrow B$
$W \rightarrow Z$	Nothing
$A \rightarrow aA$	$A \rightarrow a$
$A \rightarrow bA$	$A \rightarrow b$
$B \rightarrow Ba$	$B \rightarrow a$
$B \rightarrow Bb$	$B \rightarrow b$

Sehingga CFG tanpa nullable nya

$$S \rightarrow XY$$
$$X \rightarrow Zb \mid b$$
$$Y \rightarrow bW \mid b$$
$$Z \rightarrow AB \mid A \mid B$$
$$W \rightarrow Z$$
$$A \rightarrow aA \mid bA \mid a \mid b$$
$$B \rightarrow Ba \mid Bb \mid a \mid b$$

12.2 Latihan Soal

1. Masing-masing CFG berikut memiliki produksi Ɛ, tentukan dengan CFG tanpa Ɛ

$$S \rightarrow aX \mid bX$$
$$X \rightarrow a \mid b \mid \varepsilon$$

$$S \rightarrow aX \mid bS \mid a \mid b$$
$$X \rightarrow aX \mid a \mid \varepsilon$$

$$S \rightarrow aS \mid bX$$
$$X \rightarrow aX \mid \varepsilon$$

$$S \rightarrow XaX \mid bX$$
$$X \rightarrow XaX \mid XbX \mid \varepsilon$$

2. Rubah ke bentuk CNF

$$S \rightarrow aSa \mid SSa \mid a$$

$$S \rightarrow aXX$$
$$X \rightarrow aS \mid bS \mid a$$

$$E \rightarrow E + E$$
$$E \rightarrow E * E$$
$$E \rightarrow (E)$$
$$E \rightarrow 7$$
terminal + * ( ) 7.

3. Tentukan no. 2 dengan CNF

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

Bab 13

Pushdawn Automata

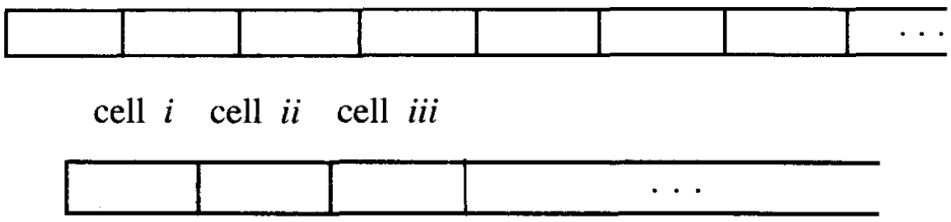
13.1 Pushdawn Automata

Setelah memperkenalkan bahasa reguler yang ditentukan oleh ekspresi reguler, kami menemukan kelas mesin abstrak (FA) dengan properti ganda berikut: Untuk setiap bahasa reguler setidaknya ada satu mesin yang berjalan dengan sukses hanya pada string input dari bahasa itu dan untuk setiap mesin membaca kumpulan kata yang diterimanya adalah bahasa reguler.

Kami mencari model matematika dari beberapa mesin yang sesuai dengan CFL; yaitu, harus ada setidaknya satu mesin yang menerima setiap CFL dan bahasa yang diterima oleh setiap mesin yang equevalen dengan bahasa bebas konteks. Kami berharap bahwa analisis mesin ini akan membantu kita memahami bahasa dalam arti yang lebih dalam, seperti sebuah analisis dari FA mengarah ke teorema bahasa reguler.

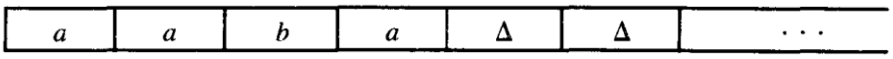
Simbul baru di Pushdown Automata

a. Simbul tape



Gambar 13.1: Simbul Tape

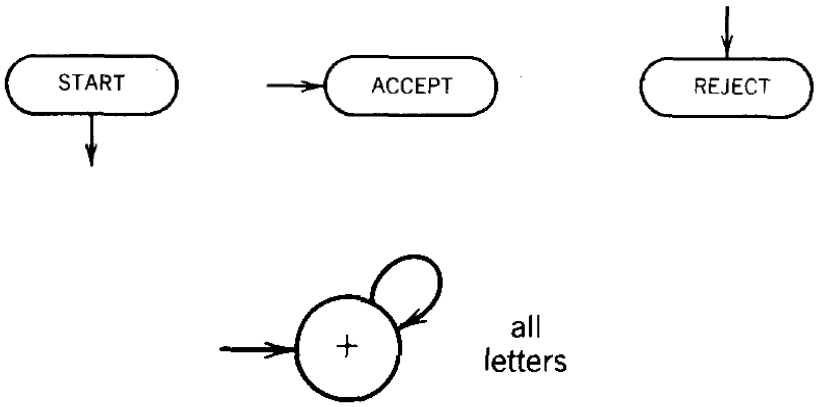
Di bawah ini kami menunjukkan contoh TAPE input yang sudah dimuat dengan masukan string aaba. Karakter "Δ" digunakan untuk menunjukkan kosong dalam TAPE sel.



Gambar 13.2: Contoh TAPE Input

Sebagian besar (semua kecuali empat) sel pada input TAPE kosong,yaitu, mereka dimuat dengan kosong, . ΔΔΔ ... Saat kami memproses TAPE ini di mesin, mesin membaca satu huruf pada satu waktu dan menghilangkan masing-masing seperti yang digunakan. Ketika mesin mencapai sel kosong pertama, mesin berhenti. selalu menganggap bahwa setelah blank pertama ditemukan, sisa TAPE juga kosong. Mesin membaca dari kiri ke kanan dan tidak pernah kembali ke sel yang telah dibaca sebelumnya.

Simbul baru untuk FA,

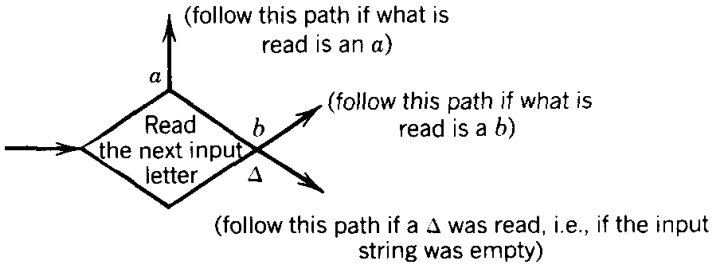


Gambar 13.3: Diagram Mesin Membaca Huruf Mencapai State Akhir

Gambar 13.4: Diagram Mesin Menbaca Semua Huruf Tidak Mencapai State Akhir

Mesin menggunakan simbul kata sifat "finalatau akhir" untuk diterapkan hanya pada state yang yang diterima oleh FA, mesin FA menyebut statusnya diterima atau ditolak yang disebut sebagai "status berhenti".

Yang paling penting dilakukan oleh state dalam FA adalah membaca huruf input dan state berikutnya menyatakan tergantung pada huruf, string atau kata yang telah dibaca. Untuk operasi ini mulai sekarang disebut status READ. Ini digambarkan sebagai kotak berbentuk berlian seperti yang ditunjukkan di bawah ini:

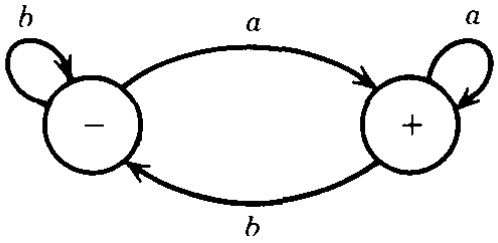


Gambar 13.5: Satus READ digambarkan Sebagai Kotak Berbentuk Berlian

Di sini arah sisi pada gambar di atas hanya menunjukkan satu dari sekian banyak kemungkinan. Ketika karakter Δ dibaca dari TAPE, itu berarti mesin sudah tidak ada lagi yang dibaca huruf input. Kemudian mesin selesai memproses string masukan. Sisi huruf Δ akan mengarah ke ACCEPT jika status mesin telah berhenti Di state akhir dan untuk MENOLAK jika pemrosesan berhenti dalam keadaan bukan state akhir. Dalam diagram Finite automata yang lama untuk FA, mesin tidak pernah tidak digambarnya sebagaimana diagram seperti ini ketika membaca huruf input. Dalam diagram baru ini mengenali dengan membaca huruf blank dari TAPE.

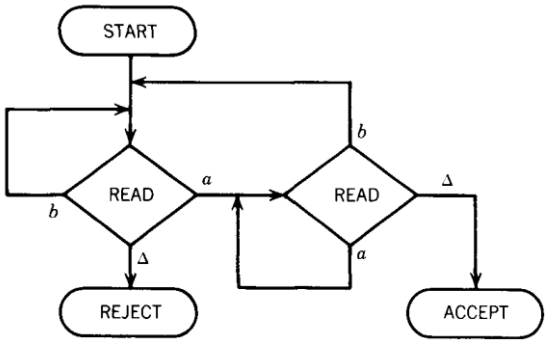
Contoh gambar yang baru sebagaimana berikut:

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

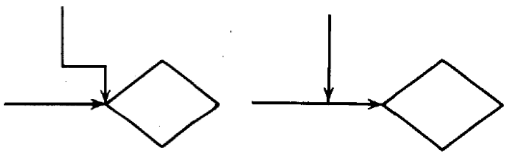


Menjadi

(FA menerima semua kata yang berakhiran dengan huruf a) menjadi, menjadi simbul baru mesin di bawah ini:

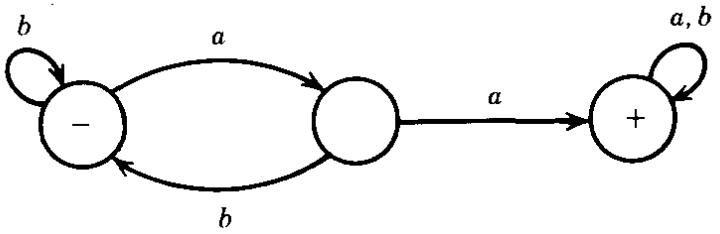


Atau

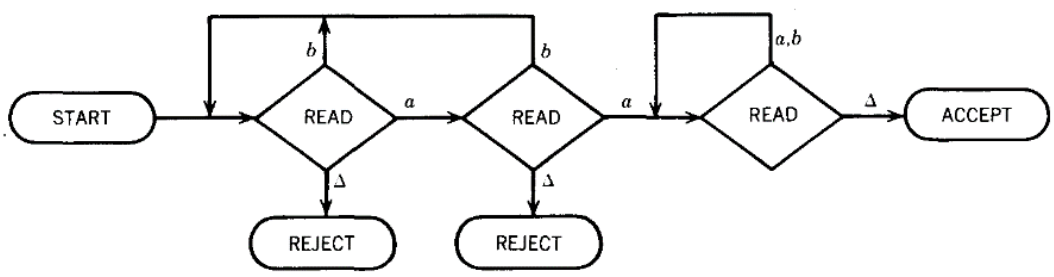


Contoh lain dari notasi bergambar baru:

Contoh



Menjadi



Gambar-gambar ini lebih mirip "bagan alur" yang kita kenal daripada gambar lama untuk FA lakukan. Studi umum tentang flowchart sebagai matematis terstruktur adalah bagian dari Teori Komputer. Alasannya membuat diagram baru untuk FA agar lebih mudah dipaham hal ini mesin FA dapat disebut PUSHDOWN STACK atau PUSHDOWN STORE.

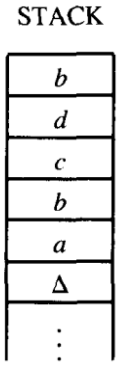
Ini adalah konsep yang mungkin dalam mempelajari tentang Struktur Data. PUSHDOWN STACK adalah tempat memasukkan huruf (atau informasi lainnya) yang dapat disimpan sampai dengan memasukkan huruf yang lain lagi(huruf sebanyak yang kita inginkan). Operasi PUSH menambahkan huruf ke dalam stack yang kosong. Kemudian huruf baru ditempatkan di paling atas baris di dalam stack,dan seterusnya, Sebelum mesin mulai memproses string input, STACK dianggap kosong, yang artinya setiap lokasi penyimpanan di dalamnya awalnya berisi blank. Jika STACK kemudian diberi makan huruf a, b, c, d dengan urutan instruksi ini maka hal ini dapat dilihat sebagaimana berikut

PUSH *a*
PUSH *b*
PUSH *c*
PUSH *d*

maka huruf teratas dalam STACK adalah d, yang kedua adalah c, yang ketiga adalah b, dan keempat adalah a. Jika sekarang kita menjalankan instruksi:

PUSH *b*

huruf b akan ditambahkan ke STACK di atas. d akan didorong turun ke posisi 2, c ke posisi 3, yang lain b ke posisi 4, dan bawah a ke posisi 5. Satu representasi bergambar dari STACK dengan huruf-huruf ini di dalamnya ditampilkan di bawah. Di bawah bagian bawah a kami menganggap bahwa sisa STACK, yang,seperti INPUT TAPE, memiliki banyak lokasi penyimpanan yang tak terhingga, hanya menampung kosong.

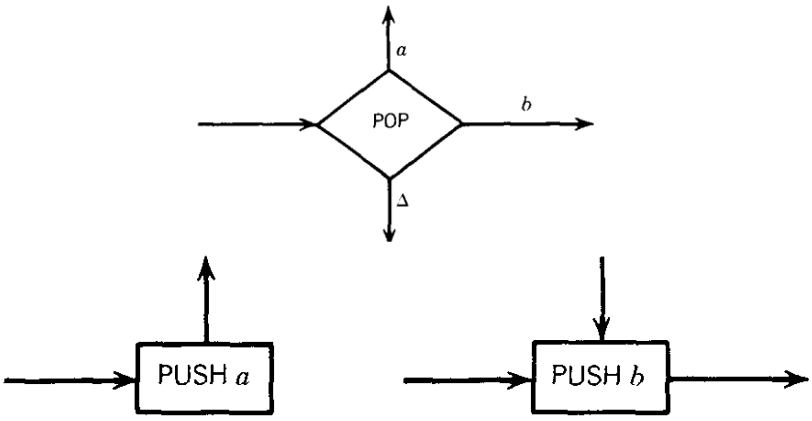


Instruksi untuk mengeluarkan huruf dari STACK disebut POP. Ini menyebabkan huruf di atas STACK dikeluarkan dari STACK.

Huruf-huruf lainnya masing-masing dipindahkan ke atas satu lokasi. Sebuah PUSHDOWN STACK disebut LIFO yang berarti huruf "TERAKHIR yang masuk ke stack dikeluarkan dari stack. Tidak seperti tempat penyimpanan komputer biasa, yang memungkinkan akses acak (kita dapat mengambil barang dari mana saja terlepas dari urutan pemberiannya). TUMPUKAN PUSHDOWN kita baca hanya huruf teratas. Jika kita ingin membaca huruf ketiga di STACK kita harus ke perintah POP, POP, POP.

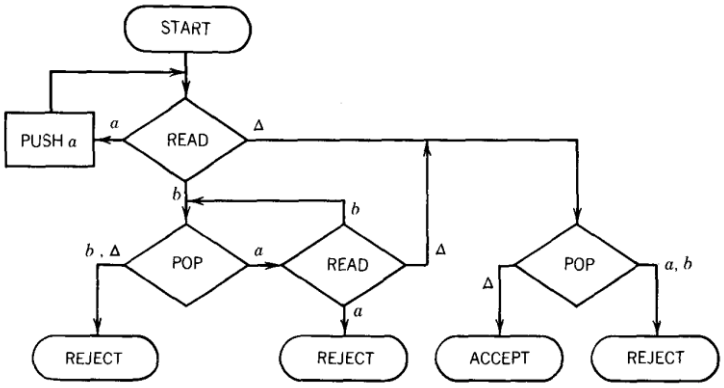
Kita dapat menambahkan PUSHDOWN STACK dan operasi PUSH dan POP maka gambar baru FA sebagaimana berikut:

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress



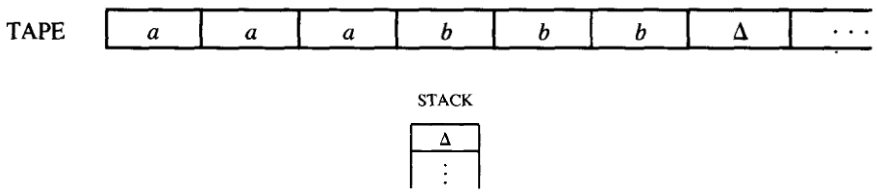
CONTOH

Perhatikan PDA berikut ini:

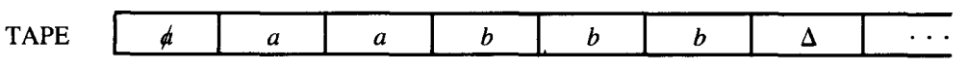


Gambar 13.6: PDA

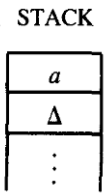
Sebelum memulai menganalisis mesin ini secara umum, mari kita lihat cara kerjanya pada string input aaabbb. Kita mulai dengan mengasumsikan bahwa string ini telah didalam TAPE. Dan selalu memulai pengoperasian PDA dengan STACK kosong seperti yang ditunjukkan:



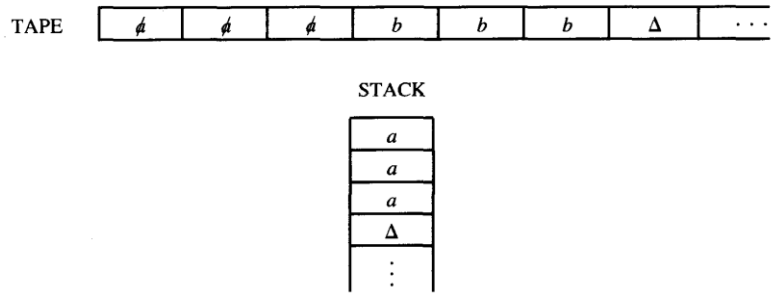
Kita mulai dari start. Dari sana kami melanjutkan langsung sisi arah ke atas kiri READ a, state yang membaca huruf a yang pertama dari input string ayang ada pada tape.



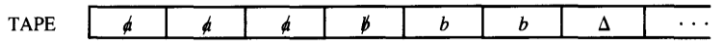
Setelah Read a maka sisi menuju ke perintah Push a, dengan demikian stack menjadi,



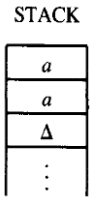
Sisi kotak PUSH membawa kembali ke garis yang ke kotak READ a yang sama, jadi kembali ke state ini. Sebanyak tiga kali reada kemudian menuju kotak Push tiga kali maka stack menjadi diagram sebagai berikut,



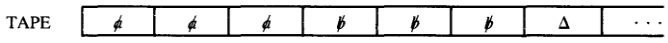
Setelah PUSH ketiga a, kita diarahkan kembali ke state READ yang sama lagi. Namun kali ini, kita membaca huruf b. Ini berarti bahwa kita mengambil sisi b keluar dari state ini ke POP kiri bawah. Membaca b meninggalkan TAPE seperti ini:



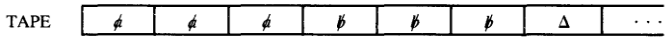
Di bagian state POP mengambil elemen huruf a teratas dari STACK.



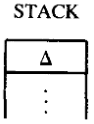
READ b mengikuti jalur arah panah POP a



Baris dari POP membawa kita lagi ke READ yang sama. Ada satu huruf tersisa pada input TAPE, kemudian membacanya dan membaca TAPE kosong, yaitu, semua kosong. Namun, mesin belum mengetahui bahwa TAPE kosong. Ia akan menemukan ini hanya ketika selanjutnya mencoba membaca TAPE dan menemukan tape kosong Δ.



Dari Read b yang baru saja dibaca oleh mesin akan dikembalikan ke state POP. Kami kemudian mengambil a terakhir dari STACK, sehingga stack menjdi diagram sebagai berikut,

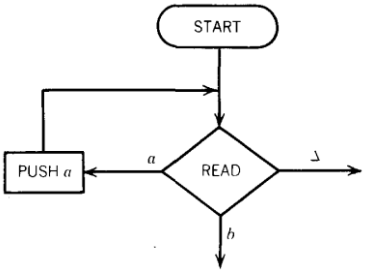


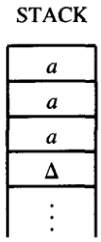
Artinya bahwa ketika mesin PDA operesi Read a tiga kali diikuti perintah dorong (PUSH) tiga kali maka isi Stacknya adalah aaa, kemudian membaca Read b tiga kali kemudian juga dorong (POP) atau keluarkan aaa secara berganian tiga kali sehingga kondisi stack seperti semula kembali stack kosong. Oleh karena itu, kata aaabbb yang diterima oleh mesin ini.

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

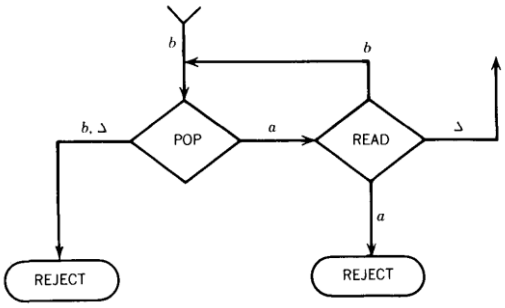
Mari kita lihat mengapa

Bagian pertama dari mesin adalah rangkaian State yang membaca dari TAPE beberapa huruf a berturut-turut dan mendorong mereka ke dalam STACK.



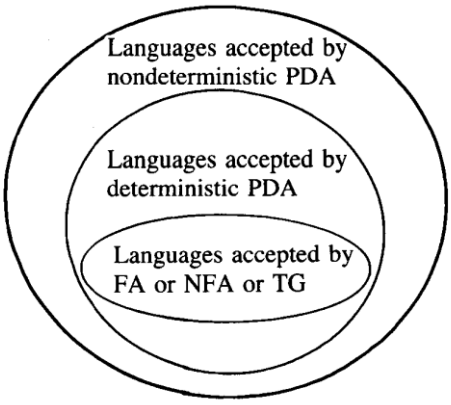


Mesin membaca Read bbb berturut –turut diikuti dengan dorongan (POP a) juga berturut turut hingga membaca Δ mencapai state berikutnya,akan tetapi bukan membaca Read a, jika membaca Read maka mesin akan berhenti (Reject), jika berarkhir pada state akhir mengikuri jalur mendorong Δ, maka aaabbb kata dari bahasa yang diterima oleh PDA, dengan demikian maka dapat disimpulkan bahawa mesin PDA ini mmerima bahasa :



$\{a^n b^n, \quad n = 0 \quad 1 \quad 2 \quad 3 \ldots\}$

Diagram Venn berikut menunjukkan relasi dari ketiga jenis mesin PDA



Gambar 13.7: Diagram Venn

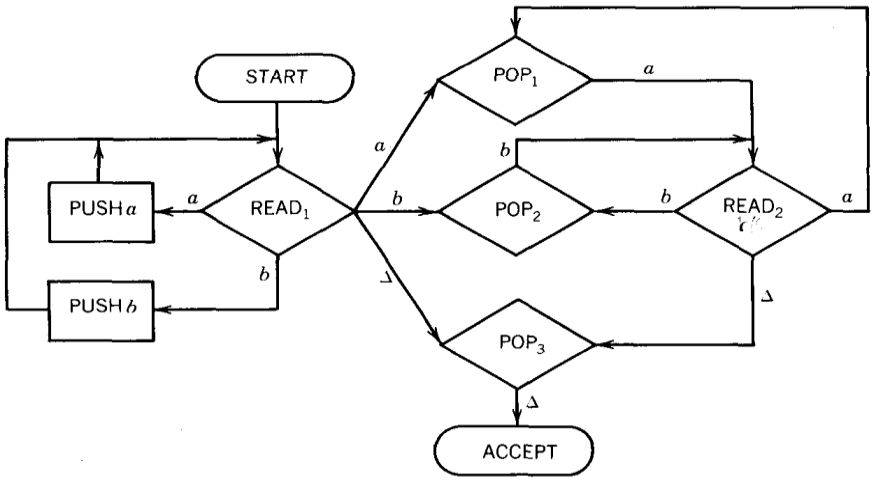
CONTOH

Ingat bahasanya:

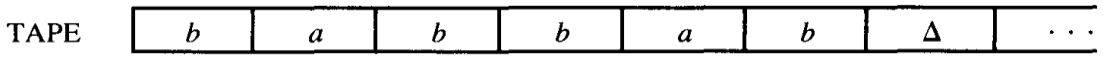
EVENPALINDROME = {S reverse dari S di mana S di dalam (a + b)*}

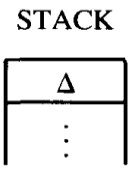
= { Ɛ,aa, bb, aaaa, abba, baab, bbbbaaaaaa... }

Ini adalah bahasa semua palindrom dengan jumlah huruf genap. Satu mesin untuk menerima bahasa ini digambarkan di bawah ini:



Kita perintahkan mesin membaca babbab, maka kondisi tape dan stack sebagai berikut





Kita dapat melacak jalur dimana input ini dapat diterima secara berturut-turut Sebagaimana pada tabel di bawah ini:

STATE	STACK	TAPE
START	Δ · · ·	*babbab*Δ · · ·
$READ_1$	Δ · · ·	¢*abbab*Δ · · ·
PUSH *b*	*b*Δ · · ·	¢*abbab*Δ · · ·
$READ_1$	*b*Δ · · ·	¢¢*bbab*Δ · · ·
PUSH *a*	*ab*Δ · · ·	¢¢*bbab*Δ · · ·
$READ_1$	*ab*Δ · · ·	¢¢¢*bab*Δ · · ·
PUSH *b*	*bab*Δ · · ·	¢¢¢*bab*Δ · · ·
$READ_1$	*bab*Δ · · ·	¢¢¢¢*ab*Δ · · ·

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

Jika kita akan menerima string input ini, di sini harus membuat dari jalur atau sirkuit kiri ke sirkuit kanan:

POP_2	$ab\Delta \cdots$	$\not b\not a\not bab\Delta \cdots$
$READ_2$	$ab\Delta \cdots$	$\not b\not a\not b\not ab\Delta \cdots$
POP_1	$b\Delta \cdots$	$\not b\not a\not b\not ab\Delta \cdots$
$READ_2$	$b\Delta \cdots$	$\not b\not a\not b\not a\not b\Delta \cdots$
POP_2	$\Delta \cdots$	$\not b\not a\not b\not a\not b\Delta \cdots$
$READ_2$	$\Delta \cdots$	$\not b\not a\not b\not a\not b\Delta \cdots$

(Kami baru saja membaca bagian pertama dari banyak kosong yang tak terhingga di TAPE.)

POP_3	$\Delta \cdots$ (Popping a blank from an empty stack still leaves blanks)	$\not b\not a\not b\not a\not b\Delta\Delta \cdots$ (Reading a blank from an empty tape still leaves blanks)
ACCEPT	$\Delta \cdots$	$\not b\not a\not b\not a\not b\Delta \cdots$

Perhatikan bahwa untuk memudahkan menggambar tabel ini, kami telah memutar STACK sehingga terbaca dari kiri ke kanan bukan dari atas ke bawah. Karena ini adalah mesin nondeterministik, ada jalur lain untuk input ini bisa saja diambil. Namun, tidak satupun dari mereka mengarah pada penerimaan. Di bawah ini kami melacak jalur yang gagal.

STATE	STACK	TAPE
START	Δ	*babbab*
$READ_1$ (We had no choice but to go here)	Δ	*babbab*
PUSH *b* (We could have chosen to go to POP_2 instead)	*b* (We know there are infinitely many blanks underneath this *b*)	*babbab* (Notice that the TAPE remains unchanged except by READ statements)
$READ_1$ (We had no choice but to go here from PUSH *b*	*b*	*babbab*
POP_1 (Here we exercised bad judgment and made a poor choice, PUSH *a* would have been better)	Δ (When we pop the *b*, what is left is all Δ's)	*babbab*
CRASH (This means that when we were in POP_1 and found a *b* on top of the STACK we tried to take the *b*-edge out of POP_1. However, there is no *b*-edge out of POP_1.)		

Pendekatan lain yang gagal untuk menerima input babbab adalah dengan mengulang sekitar sirkuit READ 1-->PUSH enam kali sampai seluruh string telah didorong ke STACK. Setelah ini, A akan dibaca dari TAPE dan kita harus pergi ke POP3. POP ini akan menanyakan apakah STACK kosong. Tidak akan menjadi, sehingga jalan akan CRASH di sini. Kata A diterima oleh mesin ini melalui urutan:

START ---> READ --> POP 3 --> ACCEPT

Seperti di atas, kita tidak akan memasukkan semua elips (. . .) ke dalam tabel yang mewakili jejak. Kami memahami bahwa TAPE memiliki banyak kosong yang tak terhingga tanpa harus menulis:

babbabΔ...

Seperti yang akan kita lihat dalam Teorema 39, PDA deterministik tidak menerima semua bahasa bebas konteks. Mesin yang kita butuhkan untuk tujuan kita adalah nondeterministik otomatisasi push-down. Kami akan menyebut mesin ini PDA dan hanya gunakan DPDA (otomat pushdown deterministik) pada yang khusus (Bab 21). Tidak perlu untuk singkatan NPDA, lebih dari yang ada untuk NTG (TG nondeterministik).

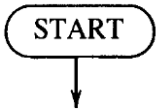
Dalam membangun mesin baru kami, kami harus membuat beberapa arsitektur keputusan. Haruskah kita menyertakan perangkat memori?-ya. Haruskah itu menjadi tumpukan (stack), antrian atau random akses ? stack. Satu stack atau lebih?--satu. Deterministik?- tidak. Sangat banyak state nya. Bisakah kita menulis di INPUT

TAPE?- tidak. Bisakah kita membaca ulang inputnya?-tidak. Ingat kita tidak mencoba untuk menemukan struktur yang terbentuk secara alami; kami adalah pembuat yang mencoba menemukan mesin yang mengenali CFL. Ujian apakah keputusan kita benar akan datang di bab berikutnya. Kami sekarang dapat memberikan definisi lengkap dari PDA.

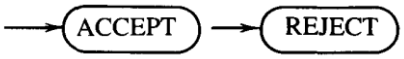
Definisi Grafis PDA

Sebuah robot pushdown, PDA, adalah kumpulan dari delapan hal:

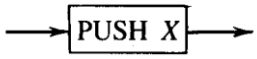
1) Alfabet £ dari huruf input.
2) Sebuah input TAPE (tak terbatas dalam satu arah). Awalnya string input huruf ditempatkan pada TAPE mulai dari sel i. sisa TAPE kosong.
3) Alfabet F dari karakter STACK.
4) Sebuah STACK pushdown (tak terbatas dalam satu arah). Awalnya STACK adalah kosong (berisi semua kosong).
5) Satu status MULAI yang hanya memiliki tepi luar, tanpa tepi dalam.



6) Status penghentian ada dua jenis: beberapa MENERIMA dan beberapa REJECT. Mereka punya di-tepi dan tidak ada di luar



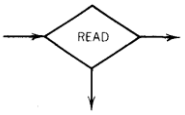
7) Sangat State PUSH karakter ke STACK. Berbentuk



di mana X adalah sembarang huruf di F.

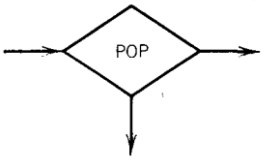
8) Banyak State percabangan dari dua jenis:

State-state yang membaca huruf tidak digunakan berikutnya dari TAPE



yang mungkin memiliki sisi luar yang diberi label dengan huruf dari Alphabet dan yang kosong karakter nullable, tanpa batasan duplikasi label dan tidak ada desakan bahwa ada label untuk setiap huruf Alphabet., atau kosong.

State yang membaca karakter teratas STACK yang didorong keluar



Untuk menjalankan string huruf input pada PDA berarti mulai dari start dan ikuti sisi yang tidak berlabel dan sisi yang berlabel (membuat pilihan sisi bila perlu) untuk menghasilkan jalur. Jalur ini akan berakhir pada state berhenti atau akan mesin macet dalam state bercabang ketika tidak ada sisi yang sesuai dengan huruf/karakter yang dibaca ada. Kapan huruf dibaca dari TAPE atau huruf yang ada dari STACK.

String input dengan jalur yang berakhir di ACCEPT dikatakan diterima. String input yang dapat mengikuti pilihan jalur dikatakan diterima jika setidaknya satu dari jalur ini mengarah state ACCEPT. Himpunan semua string input diterima oleh PDA disebut bahasa yang diterima oleh PDA, atau bahasa yang dikenali oleh PDA.

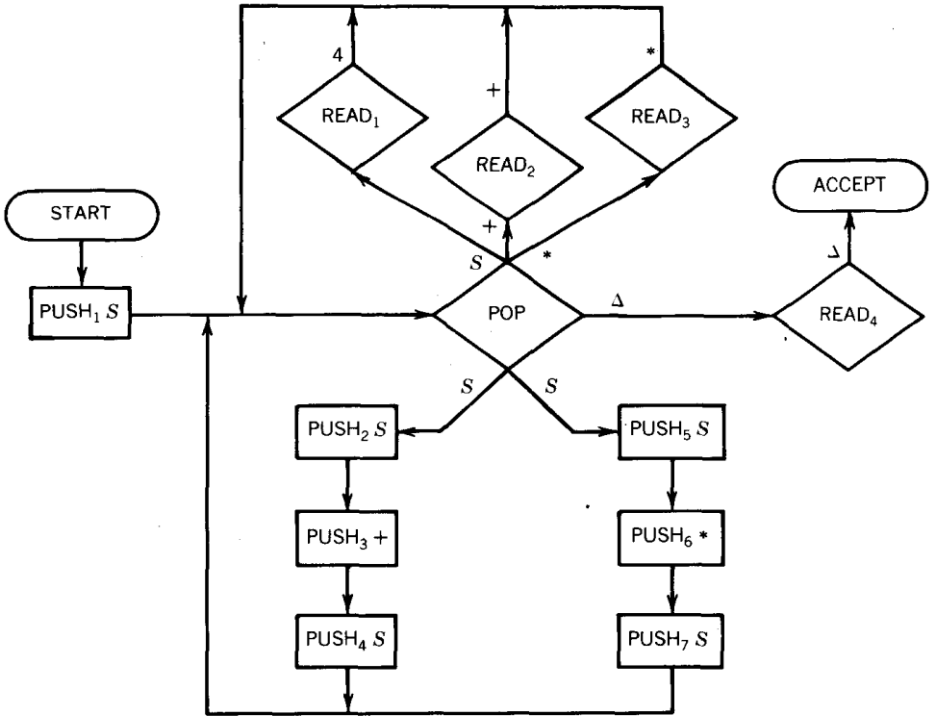
Contoh ;

Pertimbangkan bahasa yang dihasilkan oleh CFG:

$$S \rightarrow S + S \mid S * S \mid 4$$

terminalnya adalah +, *, dan 4 dan satu-satunya nonterminal adalah S.

PDA berikut menerima bahasa ini:



Kita akan menyelidiki kata yang diterima oleh PDA 4 + 4 * 4

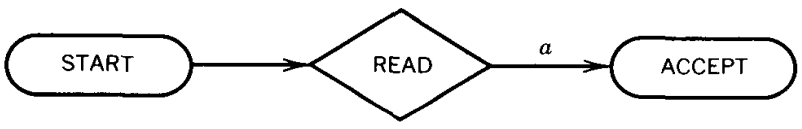
STATE	STACK	TAPE
START	Δ	4 + 4 * 4
$PUSH_1$ S	S	4 + 4 * 4
POP	Δ	4 + 4 * 4
$PUSH_2$ S	S	4 + 4 * 4
$PUSH_3$ +	+ S	4 + 4 * 4
$PUSH_4$ S	S + S	4 + 4 * 4
POP	+ S	4 + 4 * 4
$READ_1$	+ S	+ 4 * 4
POP	S	+ 4 * 4
$READ_2$	S	4 * 4
POP	Δ	4 * 4
$PUSH_5$ S	S	4 * 4
$PUSH_6$ *	* S	4 * 4
$PUSH_7$ S	S * S	4 * 4
POP	* S	4 * 4
$READ_1$	* S	* 4
POP	S	* 4
$READ_3$	S	4
POP	Δ	4
$READ_1$	Δ	Δ
POP	Δ	Δ
$READ_4$	Δ	Δ
ACCEPT	Δ	Δ

Teorema

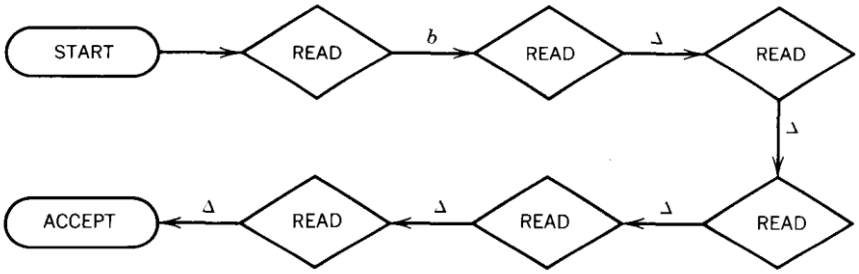
Untuk setiap bahasa reguler L ada beberapa PDA yang menerimanya.

Bukti

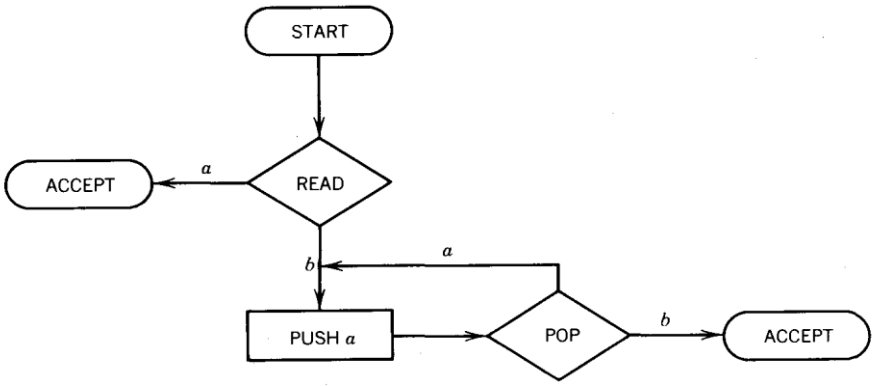
Sebenarnya sudah membahas masalah ini, tetapi kami tidak bisa secara resmi buktikan apa pun sampai kami menetapkan definisi PDA. Karena L regular, itu diterima oleh beberapa FA. Algoritma konstruktif untuk mengubah FA menjadi PDA yang setara disajikan di awal dari bab ini. Satu perbedaan penting antara PDA dan FA adalah panjang jalur yang dibentuk oleh input yang diberikan. Jika string tujuh huruf dimasukkan ke dalam FA, itu mengikuti jalan tepat tujuh sisi panjangnya. Di PDA, jalurnya bisa lebih panjang atau lebih pendek. PDA di bawah ini menerima bahasa reguler yaitu "semua kata yang dimulai dengan a". Tapi tidak peduli berapa lama string input, jalurnya hanya satu atau proses penerimaan dua sisi yang panjang.



Karena kita dapat terus memproses bagian yang kosong pada TAPE bahkan setelah semua huruf input telah dibaca, kita dapat memiliki jalur yang panjang atau bahkan tak terbatas disebabkan oleh kata-kata masukan yang sangat pendek. Misalnya, PDA berikut menerima: hanya kata b, tetapi harus mengikuti jalan tujuh sisi menuju penerimaan:



Mesin berikut menerima semua kata yang dimulai dengan a di jalur dua sisi dan loop selamanya pada input apa pun yang dimulai dengan a b. (Kita dapat mempertimbangkan ini jalan tak terbatas jika kita menginginkannya.)



Kita akan lebih ingin tahu tentang konsekuensi dari jalan tak terbatas nanti.

Hasil berikut akan membantu kita di bab berikutnya.

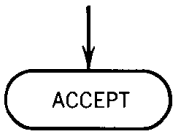
Teorema

Diberikan PDA apapun, ada PDA lain yang menerima bahasa yang sama persis dengan pilihan tambahan bahwa setiap kali jalan mengarah ke ACCEPT the STACK dan TAPE hanya berisi sell kosong (blank).

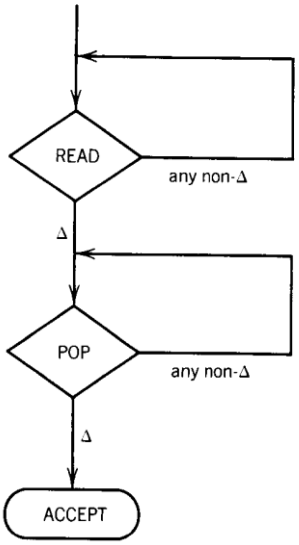
Bukti

Kami menyajikan algoritma konstruktif yang akan mengubah PDA apa pun menjadi PDA dengan pilihan yang disebutkan.

Setiap kali kita memiliki mesin:



kita ganti dengan diagram dibawah ini :



Pushdown Automata (PDA) merupakan sebuah teknik pengujian kalimat/string menggunakan pendekatan stack PDA terdiri atas pasangan 7 buah tuple

$$M = (Q, \Sigma, \Gamma, q0, Z0, \delta, A),$$

dimana:

Q : himpunan hingga state,

Σ : alfabet input,

Γ : alfabet stack,

$q0 \in Q$: stata awal,

$Z0 \in \Gamma$: simbol awal stack, dan

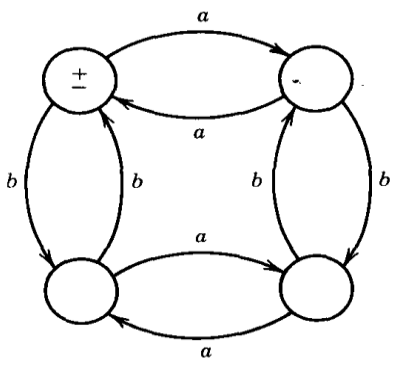
$A \subseteq Q$: himpunan stata penerima

Contoh kasus: Deterministic PDA

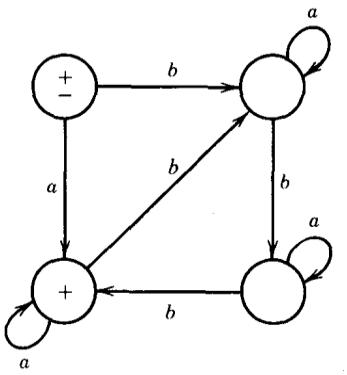
Jika diketahui sebuah PDA M = (Q, Σ, Γ, q0, Z0, δ, A) merupakan sebuah PDA deterministik untuk pengujian palindrome memiliki tuple sebagai berikut. Q = {q0, q1, q2 }, A = {q2}, Σ = {a, b, c}, Γ = {a, b, Z0}, dan fungsi transisi δ terdefinisi melalui tabel beriku.

13.2 Latihan Soal

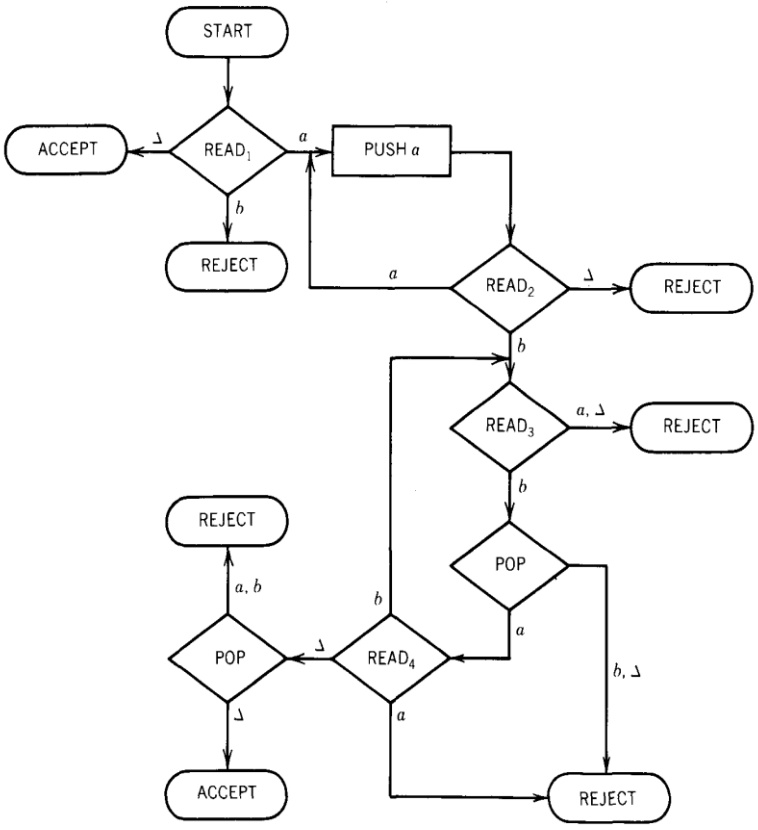
Ubah FA berikut menjadi PDA yang setara:

1. 

2.



3. Perhatikan PDA deterministik berikut:



Tunjukkan apa yang terjadi INPUT TAPE dan STACK saat masing-masing kata berikut ini.

a) abb
b) abab
c) abb
d) aabbbb.

.

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

Bab 14

Mesin Turing

14.1 Mesin Turing

Mesin Turing, pertama kali dijelaskan oleh Alan Turing di Turing 1936-197, adalah perangkat komputasi abstrak sederhana yang dimaksudkan untuk membantu menyelidiki tingkat dan keterbatasan dari apa yang dapat dihitung. 'Mesin Automaton' Turing, sebagaimana ia menyebutnya pada tahun 1936, secara khusus dirancang untuk menghitung bilangan real. Mereka pertama kali diberi nama 'Mesin Turing' oleh Gereja Alonzo dalam ulasan makalah Turing (Gereja 1937). Hari ini, mereka dianggap sebagai salah satu model dasar komputabilitas dan (teoretis) ilmu komputer.

14.2 Dampak Mesin Turing pada Ilmu Komputer

Alan Turing saat ini adalah salah satu tokoh ilmu komputer yang paling terkenal. Banyak yang menganggapnya sebagai bapak ilmu komputer dan fakta bahwa penghargaan utama dalam komunitas ilmu komputer yang disebut penghargaan Turing adalah indikasi yang jelas akan hal itu (Daylight 2015). Hal ini diperkuat dengan perayaan seratus tahun Turing dari tahun 2012, yang sebagian besar dikoordinir oleh S. Barry Cooper. Hal ini menghasilkan tidak hanya sejumlah besar peristiwa ilmiah di sekitar Turing tetapi juga sejumlah inisiatif yang membawa gagasan Turing sebagai bapak ilmu komputer juga ke publik yang lebih luas (Bullynck, Daylight, & De Mol 2015). Di antara kontribusi Turing yang saat ini dianggap sebagai perintis, makalah tahun 1936 tentang mesin Turing menonjol sebagai salah satu yang memiliki dampak terbesar pada ilmu komputer. Argumen terkait lainnya adalah bahwa Turing adalah orang pertama yang "menangkap" ide mesin serba guna melalui gagasannya tentang mesin universal dan dalam pengertian ini dia juga "menemukan" komputer modern (Copeland & Proudfoot 2011). Namun, penelitian sejarah baru-baru ini juga menunjukkan bahwa seseorang harus memperlakukan dampak mesin Turing dengan sangat hati-hati dan bahwa seseorang harus berhati-hati dalam menyesuaikan masa lalu ke masa kini.(Mol and Liesbeth 2018)

Saat ini juga, mesin Turing dan teorinya merupakan bagian dari landasan teoretis ilmu komputer. Ini adalah referensi standar dalam penelitian tentang pertanyaan mendasar seperti:

Apa itu algoritma?

Apa itu perhitungan?

Apa itu perhitungan fisik?

Apa itu perhitungan yang efisien?

Dan seterusnya.

14.3 Definisi Mesin Turing

Turing memperkenalkan mesin Turing dalam konteks penelitian dasar-dasar matematika. Lebih khusus lagi, ia menggunakan perangkat abstrak ini untuk membuktikan bahwa tidak ada metode atau prosedur umum yang efektif untuk memecahkan, menghitung, atau menghitung setiap contoh dari masalah berikut:

Entscheidungsproblem Masalah untuk memutuskan setiap pernyataan dalam logika orde pertama (yang disebut kalkulus fungsional terbatas, lihat entri pada logika klasik untuk pengantar) apakah itu diturunkan dalam logika itu atau tidak.

Perhatikan bahwa dalam bentuk aslinya (Hilbert & Ackermann 1928), masalahnya dinyatakan dalam hal validitas daripada derivabilitas. Mengingat teorema kelengkapan Gödel (Gödel 1929) membuktikan bahwa ada prosedur yang efektif (atau tidak) untuk derivabilitas juga merupakan solusi untuk masalah dalam bentuk validitasnya. Untuk mengatasi masalah ini, seseorang memerlukan gagasan formal tentang "prosedur efektif" dan mesin Turing dimaksudkan untuk melakukan hal itu.

Mesin Turing kemudian, atau mesin komputasi seperti yang disebut Turing, dalam definisi asli Turing adalah mesin yang mampu melakukan serangkaian konfigurasi terbatas q1,……,qn

(status mesin, disebut m-configurations oleh Turing). Ini dilengkapi dengan tape (pita) satu arah tak terbatas dan satu dimensi yang dibagi menjadi kotak yang masing-masing mampu membawa tepat satu simbol. Setiap saat, mesin memindai isi satu persegi r yang kosong (dilambangkan dengan S0) atau berisi simbol S1,…..…,Sm dengan S1=0 dan S2=1

Mesin adalah mesin automata berarti bahwa pada saat tertentu, perilaku mesin sepenuhnya ditentukan oleh state dan simbol saat ini. Mesin ini dikontraskan dengan apa yang disebut mesin pilihan State selanjutnya bergantung pada keputusan perangkat atau operator eksternal (Turing 1936–7: 232). Mesin Turing mampu melakukan tiga jenis aksi:

Mesin Turing menemukan aplikasi dalam teori informasi algoritmik dan studi kompleksitas, pengujian perangkat lunak, komputasi kinerja tinggi, pembelajaran mesin, rekayasa perangkat lunak, jaringan komputer, dan komputasi evolusioner

Karena mesin Turing menjadi model utama sistem komputer, mesin turing akan memiliki kemampuan output. Output sangat penting, sehingga sebuah program tanpa pernyataan output mungkin tidak berguna sama sekali, Perhatikan program berikut :

1. READ X

2. IF X = 1 THEN END

3. IF X = 2 THEN DIVIDE X BY 0

4. IF X > 2 THEN GOTO STATEMENT 4

Mari kita asumsikan bahwa inputnya adalah bilangan bulat positif. Jika program berakhir secara alami, maka kita tahu X adalah 1. Jika diakhiri dengan membuat overflow atau terganggu oleh beberapa pesan kesalahan peringatan perhitungan ilegal (crash), maka kita tahu bahwa X adalah 2. Jika ternyata program kita dihentikan karena melebihi waktu yang kita tentukan di komputer, maka kita tahu X adalah lebih besar dari 2. Kita akan melihat sebentar lagi bahwa trikotomi yang sama berlaku pada mesin Turing.

14.4 Definisi

Pembicaraan tentang "Type" dan "Head baca-tulis" dimaksudkan untuk membantu (dan mengungkapkan sesuatu tentang waktu di mana Turing sedang menulis) tetapi tidak memainkan peran penting dalam definisi mesin Turing.(Utdirartatmo 2001) Dalam situasi di mana secara formal mesin Turing diperlukan, adalah tepat untuk menguraikan definisi mesin dan program dalam matematis. Secara formal, mesin Turing dapat d sebagai quadruple M =(Q,Σ,s,δ)

di mana:

Q adalah himpunan berhingga state q

∑ adalah himpunan berhingga dari simbol

S adalah state awal s∈Q

∂ adalah fungsi transisi yang menentukan langkah selanjutnya:

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

$$:(Q\times\Sigma)\rightarrow(\Sigma\times\{L,R\}\times Q)$$

Fungsi transisi untuk mesin M adalah fungsi dari state komputasi ke state komputasi. Jika $\partial(qi,Sj)=(Sij,D,qij)$, maka ketika state mesin adalah qj, membaca simbol Sj, M menggantikan Sj dengan Sij, bergerak ke arah D $\in\{L,R\}$ dan menuju ke state qij. Menggambarkan Perilaku Mesin Turing.

14.5 Perilaku Mesin Turing

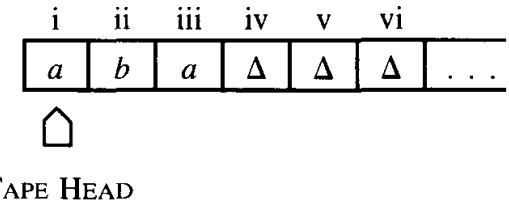
Kami memperkenalkan representasi yang memungkinkan kami untuk menggambarkan perilaku atau dinamika mesin Turing Mn , mengandalkan notasi konfigurasi lengkap (Turing 1936–7:232) yang sekarang juga dikenal sebagai deskripsi instan (ID) (Davis 1982: 6). Pada setiap tahap perhitungan Mi ID-nya diberikan oleh:

(1) Isi tape, yaitu kata datanya

(2) Lokasi Head pembacaan

(3) State internal mesin

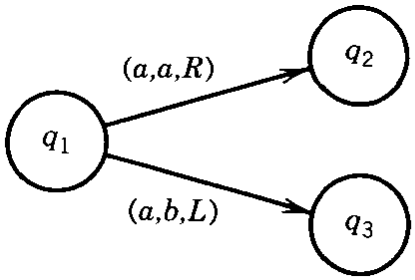
Jadi, diberikan beberapa mesin Turing M yang dalam state qi memindai simbol Sj, ID-nya diberikan oleh Pqi Sj Q di mana P dan Q adalah kata-kata berhingga di sisi kiri dan kanan kotak yang berisi simbol Sj. Gambar 1 memberikan representasi visual dari dari mesin Turing M dalam state qi memindai rekaman itu



Gambar 14.1: Konfigurasi mesin Turing M membaca

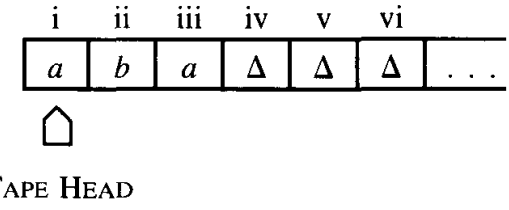
Menurut definisi, semua mesin Turing bersifat deterministik. Ini berarti ada tidak ada state q yang memiliki dua sisi atau lebih sehingga berlabel huruf sama terlebih dahulu tidak dibolehkan.

Sebagai contoh,

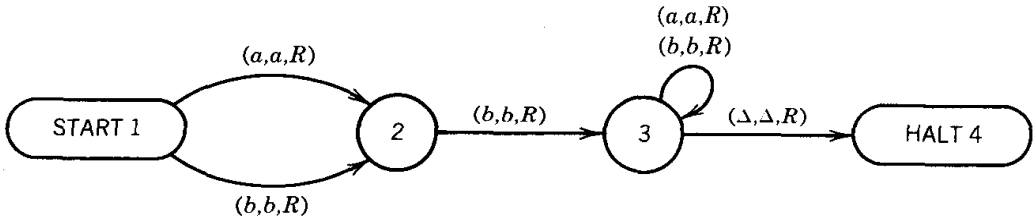


Contoh

Berikut ini adalah TAPE dari mesin Turing yang akan dijalankan pada input aba



Program untuk TM ini sebagai graf berarah dengan sisi berlabel seperti yang ditunjukkan di bawah ini



Gambar 14.2: Program untuk TM ini sebagai Graf Berarah dengan Sisi Berlabel

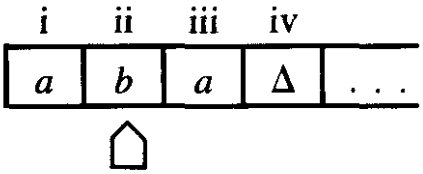
Perhatikan bahwa loop pada state 3 memiliki dua label. Sisi dari state 1 sampai state 2 dapat digambar sebagai satu sisi dengan dua label. Kita mulai, dengan sel pembacaan TAPE HEAD i dan program dalam state awal, yang di sini diberi label state 1. Kami menggambarkan ini sebagai

1

aba

Angka di atas adalah state. Di bawah ini adalah saat ini dari string pada TAPE hingga awal blank yang tak terbatas. Mungkin saja ada Δ di dalam string. Kami menggarisbawahi karakter dalam sel yang akan dibaca. Pada dalam contoh ini, KEPALA TAPE membaca huruf a dan kita ikuti sisi (a,a,R) untuk menyatakan ke state 2. Instruksi sisi ini ke TAPE HEAD adalah "membaca a, mencetak a, bergerak ke kanan."

TAPE sekarang terlihat seperti ini:



Gambar 14.3: TAPE

Perhatikan bahwa kita berhenti menulis word "TAPE HEAD" di bawah indikator TAPE. Ini masih KEPALA TAPE. Kami dapat merekam proses eksekusi dengan menulis:

$$\underset{\underline{a}ba}{1} \rightarrow \underset{a\underline{b}a}{2}$$

Pada titik ini kita berada dalam State 2. Karena kita membaca b di sel ii, kita harus naik ke State 3 di sisi berlabel (b,b,R). KEPALA PITA menggantikan b dengan b dan bergerak ke kanan satu sel. Ide untuk mengganti huurf a dengan dirinya sendiri, tetapi menyatukan struktur mesin Turing. Kami malah bisa membuat mesin yang menggunakan dua jenis instruksi: cetak atau pindahkan, tidak keduanya sekaligus.

(a, a, R) berarti bergerak, tetapi tidak mengubah sel TAPE

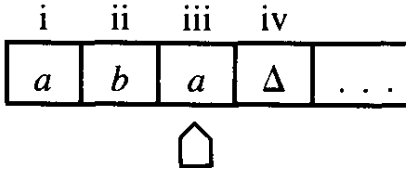
(a, b, R) berarti memindahkan dan mengubah sel TAPE

Kami sekarang sampai

$$\underset{\underline{a}ba}{1} \rightarrow \underset{a\underline{b}a}{2} \rightarrow \underset{ab\underline{a}}{3}$$

TAPE sekarang terlihat seperti ini.

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress



Kami berada di state 3 membaca a, Itu berarti kita tetap dalam state 3 tapi kita pindahkan TAPE HEAD ke sel iv.

$$\overset{3}{ab\underline{a}} \rightarrow \overset{3}{ab\underline{a}\Delta}$$

Ini adalah salah satu saat ketika kita harus menunjukkan A sebagai bagian dari isi TAPE yang bermakna.Kami sekarang di state bagian 3 membaca A, jadi kami pindah ke state 4.

$$\overset{3}{ab\underline{a}\Delta} \rightarrow \overset{4}{ab\underline{a}\Delta\Delta}$$

String input aba telah diterima oleh TM ini. Mesin ini tidak mengubah huruf apa pun pada TAPE, JADI di akhir run the TAPE masih membaca abaΔA . . . . Ini bukan persyaratan untuk penerimaan string, hanya fenomena yang terjadi. Singkatnya, seluruh eksekusi dapat digambarkan dengan eksekusi berikut:

rantai, juga disebut rantai proses, atau jejak eksekusi, atau hanya a jejak:

$$\overset{1}{\underline{a}ba} \rightarrow \overset{2}{a\underline{b}a} \rightarrow \overset{3}{ab\underline{a}} \rightarrow \overset{3}{ab\underline{a}\Delta} \rightarrow \text{HALT}$$

Mari kita pertimbangkan string input mana yang diterima oleh TM ini. Apa saja yang pertama? huruf, a atau b, akan membawa kita ke stete 2. Dari stete 2 ke stete 3 kita membutuhkan itu kita membaca huruf b. Setelah di stete 3 kami tinggal di sana saat KEPALA TAPE bergerak kanan dan kanan lagi, bergerak mungkin banyak sel sampai menemukan Δ. Kemudian kita sampai ke stete HALT dan menerima kata itu. Setiap kata yang mencapai stete 3 akhirnya akan diterima. Jika huruf kedua adalah a, maka kita menabrak state 2. Ini karena tidak ada sisi yang datang dari state 2 dengan arah untuk apa yang terjadi ketika KEPALA TAPE membaca a.

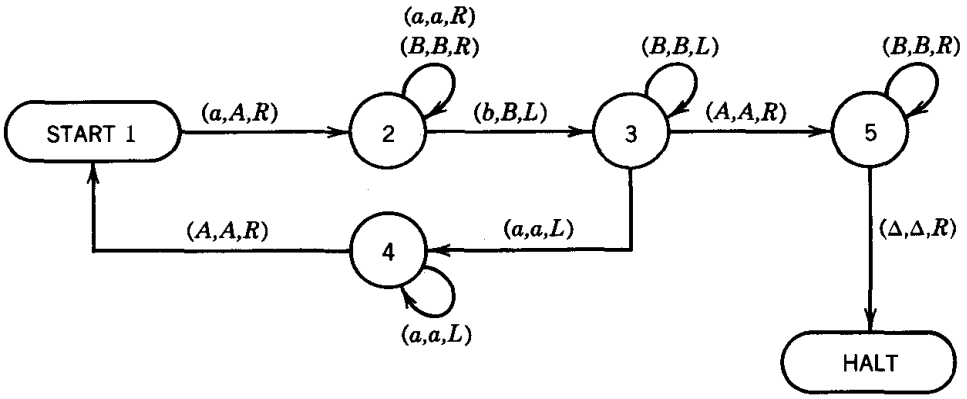
Bahasa kata yang diterima oleh mesin ini adalah: Semua kata di atas alfabet {a,b} di mana huruf kedua adalah a b. Ini adalah bahasa reguler karena juga dapat didefinisikan oleh regular ekspresi:

$$(a + b)b(a + b)^*$$

TM ini juga mengingatkan pada FA, membuat hanya satu umpan yang melewati string, menggerakkan KEPALA TAPE-nya selalu ke kanan, dan tidak pernah mengubah huruf itu telah membaca. TM dapat melakukan lebih banyak trik, seperti yang akan segera kita lihat.

Contoh

Perhatikan TM berikut.



Gambar 14.4: Contoh TM

TM, sejak dari TAPE tergantung pada huruf input. Ini adalah contoh yang lebih rumit dari sebuah TM.

Bahasa yang diterima TM ini adalah {anbn}.Dengan memeriksa program dapat melihat bahwa KEPALA TAPE dapat mencetak apapun dari huruf a, Δ atau B, atau Δ, dan dapat membaca salah satu huruf a, b, A atau B atau Blank (kosong). Secara teknis, alfabet input adalah ∑ = {a, b} dan output alfabet adalah F = {a, Δ, B}, karena Δ adalah simbol untuk sel kosong atau kosong dan bukan merupakan karakter legal dalam alfabet. Mari kita jelaskan algoritmanya,secara informal dalam bahasa Inggris, sebelum melihat grafik berarah yang merupakan programnya.

Mari kita asumsikan bahwa kita mulai dengan sebuah kata dari bahasa {anbn} di TAPE. Kita mulai dengan mengambil a di sel pertama dan mengubahnya menjadi karakter A. (Jika sel pertama tidak berisi a, program akan macet. Kita bisa atur ini dengan hanya memiliki satu tepi yang mengarah dari MULAI dan memberi label ke baca a.) Konversi dari a ke A berarti a ini telah dihitung. Kami sekarang ingin menemukan b dalam kata yang berpasangan dengan a ini. Jadi kita terus gerakkan KEPALA TAPE ke kanan, tanpa mengubah apa pun yang dilewatinya lebih, sampai mencapai yang pertama b. Ketika kita mencapai b ini, kita mengubahnya menjadi karakter B, yang sekali lagi berarti bahwa itu juga telah dihitung. Sekarang kita pindahkan KEPALA TAPE kembali ke bawah ke kiri hingga mencapai yang pertama tak terhitung sebuah. Pertama kali kami turun di TAPE, ini akan menjadi yang pertama sel ii.

Bagaimana kita tahu kapan kita sampai ke a tak terhitung pertama? Kami tidak bisa memberi tahu KEPALA TAPE untuk "menemukan sel ii." Instruksi ini tidak ada dalam repertoarnya. Kita dapat, bagaimanapun, memberi tahu KEPALA TAPE untuk terus bergerak ke kiri sampai mencapai karakter A. Ketika menyentuh A kita memantul satu sel ke kanan dan di sana kita. Dalam melakukan ini KEPALA TAPE melewati sel ii saat turun TAPE. Namun, ketika kami pertama kali di sana, kami tidak mengenalinya sebagai tujuan. Hanya ketika kita terpental dari penanda kita, A pertama ditemui, apakah kita menyadari di mana kita berada. Setengah trik dalam pemrograman TM adalah mengetahui di mana KEPALA TAPE dengan memantul dari landmark.

Ketika kami telah menemukan a paling kiri yang tak terhitung ini, kami mengubahnya menjadi A dan mulai berbaris TAPE mencari yang sesuai b. Ini berarti bahwa kita melewati beberapa a dan simbol B, yang sebelumnya kita menulis, membiarkannya tidak berubah, sampai kita sampai pada hitungan pertama b. Satu kali kami telah menemukannya, kami telah menemukan pasangan a dan b kedua kami. Kita menghitung b kedua ini dengan mengubahnya menjadi B, dan kami berbaris kembali ke TAPE mencari kami berikutnya tak terhitung a. Ini akan berada di sel iii. Sekali lagi, kita tidak bisa beri tahu KEPALA TAPE, "temukan sel iii." Kita harus memprogramnya untuk menemukan yang dimaksud sel. Instruksi yang sama seperti yang diberikan terakhir kali bekerja lagi. Kembali ke pertama A kita bertemu dan kemudian naik satu sel. Saat kami berbaris, kami berjalan melalui a B dan beberapa a sampai kita pertama kali mencapai karakter A. Ini akan menjadi yang kedua A, yang ada di sel ii. Kami memantul ini ke kanan, ke sel iii, dan menemukan sebuah. Ini kita konversi ke A dan naikkan TAPE untuk menemukan yang sesuai b.

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

Kali ini berbaris di TAPE kita kembali melewati a dan B sampai kita temukan yang pertama b. Kami mengubah ini menjadi B dan berbaris kembali mencari pertama tidak dikonversi a. Kami mengulangi proses pairing berulang-ulang.

Apa yang terjadi ketika kita telah memasangkan semua a dan b? Setelah kami mengubah b terakhir kami menjadi B dan kami bergerak ke kiri mencari a kami berikutnya menemukan bahwa setelah berbaris ke kiri melalui B terakhir, kami menemukan A. Kami menyadari bahwa ini berarti kami keluar dari sedikit a di bidang awal a di awal kata.

Kami hampir siap untuk menerima kata itu, tetapi kami ingin memastikan bahwa tidak ada lagi b yang belum dipasangkan dengan a, atau yang asing a di akhir. Oleh karena itu kami bergerak kembali melalui bidang B menjadi yakin bahwa mereka diikuti oleh yang kosong, jika tidak kata awalnya mungkin memiliki telah aaabbbb atau aaabbba.

Ketika kita tahu bahwa kita hanya memiliki A dan B di TAPE, dalam jumlah yang sama nomor, kami dapat menerima string input. Berikut ini adalah gambar dari isi TAPE pada setiap langkah dalam pemrosesan string aaabbb. Ingat, dalam jejak KEPALA TAPE adalah ditunjukkan oleh garis bawah surat yang akan dibacanya.

$$\underline{a}\ a\ a\ b\ b\ b$$

$$A\ \underline{a}\ a\ b\ b\ b$$

$$A\ a\ \underline{a}\ b\ b\ b$$

$$A\ a\ a\ \underline{b}\ b\ b$$

$$A\ a\ \underline{a}\ B\ b\ b$$

$$A\ \underline{a}\ a\ B\ b\ b$$

$$\underline{A}\ a\ a\ B\ b\ b$$

$$A\ \underline{a}\ a\ B\ b\ b$$

$$A\ A\ \underline{a}\ B\ b\ b$$

$$A\ A\ a\ \underline{B}\ b\ b$$

$$A\ A\ a\ B\ \underline{b}\ b$$

$$A\ A\ a\ \underline{B}\ B\ b$$

$$A\ A\ \underline{a}\ B\ B\ b$$

$$A\ \underline{A}\ a\ B\ B\ b$$

$$A\ A\ \underline{a}\ B\ B\ b$$

$$A\ A\ A\ \underline{B}\ B\ b$$

$A\ A\ A\ B\ \underline{B}\ b$

$A\ A\ A\ B\ B\ \underline{b}$

$A\ A\ A\ B\ \underline{B}\ B$

$A\ A\ A\ \underline{B}\ B\ B$

$A\ A\ \underline{A}\ B\ B\ B$

$A\ A\ A\ \underline{B}\ B\ B$

$A\ A\ A\ B\ \underline{B}\ B$

$A\ A\ A\ B\ B\ \underline{B}$

$A\ A\ A\ B\ B\ B\ \underline{\Delta}$

HALT

Berdasarkan algoritma ini dapat mendefinisikan satu set state yang memiliki arti sebagai berikut:

State I :

Ini adalah state awal, tetapi juga state dimana kita berada kapan pun kita beradatentang membaca tidak berpasangan terendah a. Dalam PDA kita tidak akan pernah bisa kembali kestate MULAI, tetapi dalam TM kita bisa. Ujung-ujungnya pergi dari siniharus mengubah ini menjadi karakter A dan memindahkan KEPALA TAPE ke kanan dan masukkan status 2.

State 2 :

Ini adalah state kita ketika kita baru saja mengonversi a ke A dan kami mencari yang cocok b. Kami mulai bergerak ke atas TAPE. Jika kita membaca a yang lain, kita biarkan saja dan terus berbaris TAPE, gerakkan TAPE HEAD selalu ke kanan. Jika kita membaca B, kita juga biarkan saja dan terus gerakkan KEPALA TAPE ke kanan. Kita tidak bisa membaca A sementara dalam state ini. Dalam algoritma ini semua A tetap pada kiri KEPALA TAPE setelah dicetak. Jika kita membaca A sementara kita mencari b kami dalam masalah karena kami belum memasangkan kami sebuah. Jadi kita crash. B pertama yang kita baca, jika kita cukup beruntung untuk menemukannya, adalah akhir dari pencarian di state ini. Kami mengubahnya menjadi B, pindahkan TAPE KEPALA kiri dan masukkan status 3.

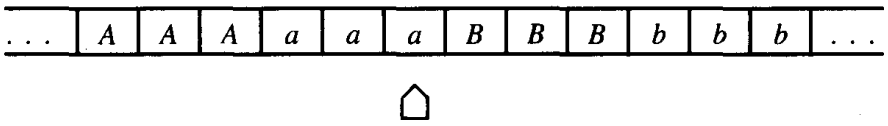
State 3 :

Ini adalah state dimana kita baru saja mengonversi a b ke B. sekarang harus berbaris ke kiri TAPE mencari bidang yang tidak berpasangan sebagai. Jika kita membaca B, kita biarkan saja dan terus bergerak ke kiri. Jika dan kapan kita membaca a, kita telah melakukan pekerjaan kita. Kita kemudian harus pergi ke state 4, yang akan mencoba menemukan yang paling kiri tidak berpasangan. Jika kita menemukan karakter b saat bergerak ke kiri, ada sesuatu yang salah dan kami harus crash. Namun, jika kita bertemu dengan karakter A sebelum kita memukul a, kita tahu bahwa pada awalnya menghabiskan kumpulan a yang tidak berpasangan dari string input dan kami mungkin siap untuk menghentikan eksekusi. Karena itu, kami meninggalkan A sendirian dan membalikkan arah ke kanan dan pindah ke state 5.

State 4 :

Kami sampai di sini ketika state 3 telah menemukan ujung paling kanan dari bidang a yang tidak berpasangan. Situasi TAPE dan TAPE HEAD terlihat seperti ini:

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress



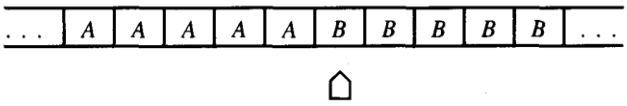
Gambar 14.5: State Bergerak ke Kiri

Dalam state ini kita harus bergerak ke kiri melalui balok padat (kita menabrak jika kita menemukan a b, a B, atau a A) sampai kita menemukan A. Ketika kita melakukannya, kita memantul ke kanan, yang membuat kita berada di paling kiri tak terhitung a.

Ini berarti bahwa kita selanjutnya harus berada di state 1 lagi.

State 5 :

Ketika kita sampai di sini pasti karena state 3 menemukan bahwa tidak ada kiri a tidak berpasangan dan itu memantulkan kita dari A paling kanan. Kita sekarang membaca B paling kiri seperti pada gambar di bawah ini:

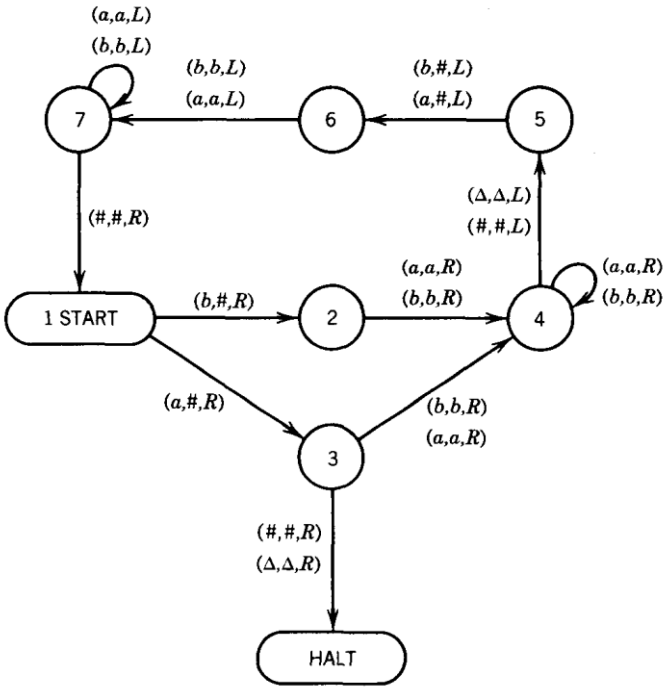


Sekarang tugas kita untuk memastikan bahwa tidak ada lagi a atau b yang tersisa di sini kata. Kami ingin memindai melalui B padat sampai kami mencapai titik kosong pertama. Karena program tidak pernah mencetak yang blank (kosong), ini akan menunjukkan akhir dari string masukan. Jika tidak ada lagi sebelum Δ, maka menerima kata dengan ke state HALT. Kalau tidak, kita crash. Untuk contoh, aabba akan menjadi AABBa dan kemudian crash karena mencari Δ kami menemukan a. sesuai dengan deskripsi di atas state untuk state dan sisi untuk sisi. Mari kita telusuri pemrosesan string input aabb dengan melihat eksekusinya sebagai berikut:

$$\underset{\underline{a}abb}{1} \rightarrow \underset{A\underline{a}bb}{2} \rightarrow \underset{Aa\underline{b}b}{2} \rightarrow \underset{A\underline{a}Bb}{3} \rightarrow \underset{\underline{A}aBb}{4} \rightarrow \underset{A\underline{a}Bb}{1}$$
$$\rightarrow \underset{AA\underline{B}b}{2} \rightarrow \underset{AAB\underline{b}}{2} \rightarrow \underset{AA\underline{B}B}{3} \rightarrow \underset{A\underline{A}BB}{3} \rightarrow \underset{AA\underline{B}B}{5} \rightarrow \underset{AAB\underline{B}}{5}$$
$$\rightarrow \underset{AABB\underline{\Delta}}{5} \rightarrow HALT$$

14.6 Latihan Soal

Untuk Soal 1 dan 2, pertimbangkan TM berikut:



1. Lacak rantai eksekusi string input berikut pada mesin ini.

   (i) aaa

   (ii) aba

   (iii) baba

   (iv) ababb

2. Bahasa yang diterima oleh TM ini adalah semua kata dengan bilangan ganjil huruf yang memiliki a sebagai huruf tengah. Tunjukkan bahwa ini benar dengan menjelaskan algoritma yang digunakan mesin dan arti dari setiap keadaan. Perhatikan dua bagian penting yang harus selalu ditunjukkan:

   (i) Apa pun yang memiliki a di tengah akan HALT dan

   (ii) Apa pun yang sampai ke HALT memiliki a di tengah.

3. (i)Bangun TM yang menerima bahasa semua kata yang mengandung substring bbb.

   (ii) Bangun TM yang menerima bahasa semua kata yang tidak mengandung substring bbb.

.

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

Pustaka

Brookshea, J. Glenn. 1989. *Theory of Computation: Formal Languages, Automata, and Complexity (Benjamin/Cummings Series in Computer Science)*. Redwood City: Publishing Co., Inc.

Hopcroft, John E., Jeffrey D. Ullman, Rotwani, and Rajeev Motwani. 2001. *Introduction to Automata Theory, Language, and Computation*. Addison-Wesley.

Levin, Oscar. 2009. "School of Mathematical Science University of Northern Colorado Greeley." *Mathematical Sciences Natural and Health Sciences*.

Mol, De, and Liesbeth. 2018. "Turing Machines." *Stanford Encyclopedia of Philosophy*. Retrieved (https://plato.stanford.edu/entries/turing-machine/#DefiTuriMach).

Peter, Linz. 2008. *An Introduction to Formal Languages and Automata 5th Edition*. Jones & Bartlett Publishers.

Rich, Elaine. 2008. *Automata and Complexity Theory and Applications*. Pearson Prentice Hall.

Sumarno, Sumarno, and Danang Tri Prasetiyo. 2019. "Lovebirds Type Identification Designing Based on Color Using Automaton Theory." *International Journal On Orange Technologies (IJOT)* 1(2).

Utdirartatmo, Firrar. 2001. *Teknik Kompilasi*. Yogyakarta: J&J Learning.

Biodata Penulis:

Ir. Sumarno, MM. Dilahirkan di Surabaya, 27 Mei 1961. Pada tahun 1991, penulis mendapatkan gelar Sarjana teknik elektro dari Universitas Muhammadiyah Suranaya dan melanjutkan jenjang Sarjana S2 Di Universitas Muhammadiyah Malang lulus tahun 2008, sejak tahun 1993, terdaftar sebagai Dosen tetap di Universitas Muhammadiyah Sidoarjo, mengampu mata kuliah Sistem Informasi, Sistem Informasi Manajemen, Jaringan komputer dan teori Bahasa dan Automata.

Dr.Hindarto, S.Kom, MT. dilahirkan di Surabaya, 30 Juli 1973. Pada tahun 1995, penulis mendapatkan gelar Diploma dari Politeknik Negeri Surabaya dan melanjutkan jenjang Sarjana di prodi Informatika FakultasTeknik Umsida. Penulis melanjutkan magister Teknik Elektro ITS dengan program beasiswa dari DIKTI. Tahun 2007, penulis secara resmi mendakatkan gelar M.T. Penulis melanjutkan doctoral di Jaringan Cerdas Multimedia Teknik Elektro ITS dengan program beasiswa dari DIKTI. Tahun 2016, penulis secara resmi mendakatkan gelar Dr. Penulis mengawali karirnya sebagai Dosen di prodi Informatika Universitas Muhaammdiyah Sidoarjo pada Tahun 2004. Beberapa penelitian yang pernah dilakukan oleh penulis adalah tentang Deteksi Sinyal Jantung dan Deteksi Sinyal EEG.

Suhendro Busono, ST, MT, Dilahirkan tanggal 30 December 1982 penulis mendapatkan sarjana Diploma III,PENS(Politeknik Elektronika Surabaya) ITS Surabaya, Jurusan Teknik Telekomunikasi, Surabaya - Jawa Timur, Periode 2002 – 2005, Diploma IV, PENS(Politeknik Elektronika Surabaya) ITS Surabaya, Jurusan Teknik Informatika, Surabaya – Jawa Timur, Periode 2008 – 2010, dan Strata 2, Universitas Dian Nuswantoro, Jurusan Ilmu Komputer, Semarang – Jawa Tengah , Periode 2015 - 2017

Published by Umsida Press, https://press.umsida.ac.id/index.php/umsidapress

UMSIDA PRESS
Universitas Muhammadiyah Sidoarjo
Jl. Mojopahit No. 666B
Sidoarjo, Jawa Timur

ISBN 978-623-464-031-1 (PDF)