FINITE AUTOMATA, PUSHDOWN AUTOMATA, TURING MACHINE

Disusun untuk memenuhi salah satu tugas mata kuliah

Teori Bahasa Formal & Auto Mata

Dosen Pengampu : Muh. Hajar Akbar, ST., M.Kom

Disusun Oleh :

Nama : Miftahul Jannah

Nim : A1 20043

Prodi : Teknik Informatika

UNIVERSITAS NAHDLATUL ULAMA

SULAWESI TENGGARA

FINITE AUTOMATA

Finite automata adalah mesin abstrak berupa sistem model matematika dengan masukan dan keluaran diskrit yang dapat mengenali bahasa paling sederhana (bahasa reguler) dan dapat diimplementasikan secara nyata di mana sistem dapat berada di salah satu dari sejumlah berhingga konfigurasi internal disebut state. Beberapa contoh sistem dengan state berhingga antara lain pada mesin minuman otomatis atau *vending machine*, pengatur lampu

lalu lintas dan *lexical analyser*.Suatu finite automata terdiri dari beberapa bagian. Finite automata mempunyai

sekumpulan state dan aturan-aturan untuk berpindah dari state yang satu ke state yang lain, tergantung dari simbol nya. Finite automata mempunyai state awal, sekumpulan state

dan state akhir. Finite automata merupakan kumpulan dari lima elemen atau dalam bahasa matematis dapat disebut sebagai 5-tuple. Definisi formal dari finite automata dikatakan bahwa finite automata merupakan list dari 5 komponen : kumpulan state, input , aturan perpindahan, state awal, dan state akhir.

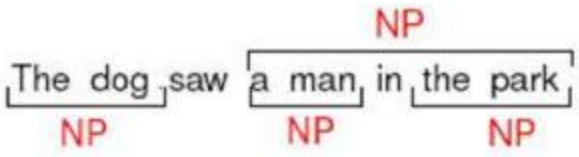
Dalam DFA sering digunakan istilah fungsi transisi untuk mendefinisikan aturan perpindahan, biasanya dinotasikan dengan _. Jika finite automata memiliki sebuah panah dari suatu state x ke suatu state y,dan memiliki label dengan simbol input 0, ini berarti bahwa, jika automata berada pada state x ketika automata tersebut membaca 0, maka automata tersebut dapat berpindah ke state y dapat diindikasikan hal yang sama dengan fungsi transisi dengan mengatakan bahwa _(x, 0) = y.

PUSHDOWN AUTOMATA

Terinspirasi dari bahasa natural manusia, ilmuwan-ilmuwan ilmu komputer yang mengembangkan bahasa pemrograman turut serta memberikan tata bahasa (pemrograman) secara formal. Tata bahasa ini diciptakan secara bebas-konteks dan disebut CFG

(*Context Free Grammar).* Hasilnya, dengan pendekatan formal ini, kompiler suatu bahasa pemrograman dapat dibuat lebih mudah dan menghindari ambiguitas ketika parsing bahasa tersebut. Contoh desain CFG untuk parser, misal : B -> BB | (B) | e untuk mengenali bahasa dengan hanya tanda kurung {‘(’,’)’} sebagai terminal-nya. Proses parsing adalah proses pembacaan untai dalam bahasa sesuai CFG tertentu, proses ini harus mematuhi aturan produksi dalam CFG tersebut.

Secara formal, CFG didefinisikan[2] : **CFG G = (V,T,P,S)**, dimana **V** adalah daftar variabel produksi **T**, adalah daftar simbol atau terminal yang dipakai dalam CFG **P**, adalah aturan produksi CFG **S**, adalah variabel start aturan produksi CFG dapat ‘dinormalkan’ dengan pola tersendiri supaya tidak ambigu dan lebih sederhana, meskipun normalisasi CFG kadang membuat aturan produksi menjadi lebih banyak dari sebelumnya. Teknik normalisasi yang digunakan dalam makalah ini adalah CNF (*Chomsky Normal For)*



Gambaran parsing bahasa natural (Inggris)

Contoh desain CNF dari bahasa CFG, semisal CFGberikut:

S -> aA | bB (1)

A-> Baa|ba

B -> bAA|ab

CFG (1) tersebut ekivalen dengan CFG dibawah ini,dimana symbol terminal memiliki variabel produksi tersendiri:

S -> DA | EB (2)

A -> BDD | ED

B -> EAA | DE D -> a

E -> b

CNF yang dihasilkan dari CFG (2) diatas ialah:

S -> DA | EB (3)

A -> BF | ED

B -> EH | DE

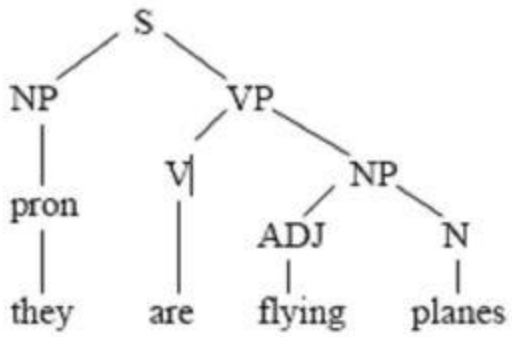
F -> DD

H -> AA

D -> a

E -> b

Setelah terbentuk CFG yang telah dinormalkan secara CNF, dalam implementasi parsing, terdapat algoritma yang berguna untuk menentukan apakah suatu untai ‘valid’, atau dapat diciptakan dari aturan-aturan CFG yang ada. Salah satu algoritma yang dapat dipakai adalah Algoritma Cocke-Younger-Kasami (CYK). Algoritma ini menyelesaikan masalah analisa kembali sebuah sub-untai yang sama karena seharusnya analisa sub-untai independen terhadap parsing sub-untai yang diparsing setelahnya. Dengan Program Dinamis, independensi yang diinginkan dapat dicapai ketika parsing. Algoritma CYK termasuk dalam bidang Program Dinamis karena algoritma ini membangun tabel status dua dimensi ketika parsing dimana penentuan parsing selanjutnya diturunkan atau dihasilkan dari parsing sebelumnya, hingga akhir untai. Selain untuk mengetahui validitas untai dalam suatu CFG, algoritma CYK yang dimodifikasi dapat dipergunakan pula untuk membangun pohon parsing.



Pohon parsing yang terbentuk dari sebuah bahasa natural

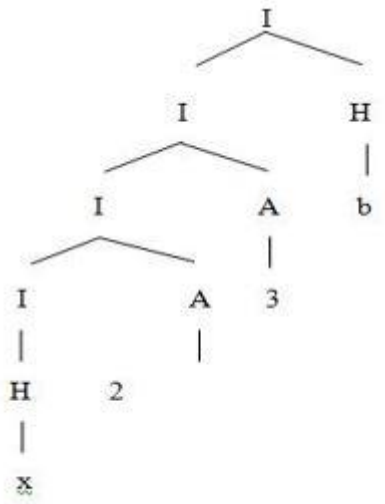
Contoh 1 :

Diketahui grammar G_1 = {I → H I H IA, H → a b c … z, A → 0 1 2 … 9} dengan I adalah simbol awal. Berikut ini kedua cara analisa sintaks untuk kalimat x23b.

Cara 1 (derivasi)

- ⇒ IH
- IAH
- IAAH
- HAAH
- xAAH
- x2AH
- x23H
- x23b

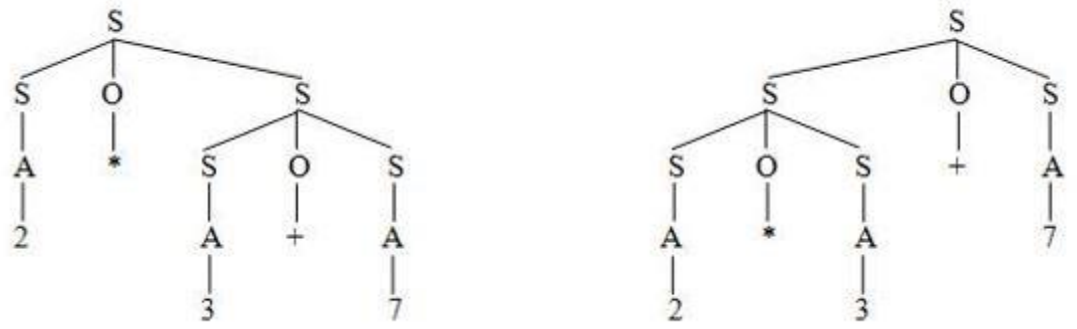
cara 2 (parsing)



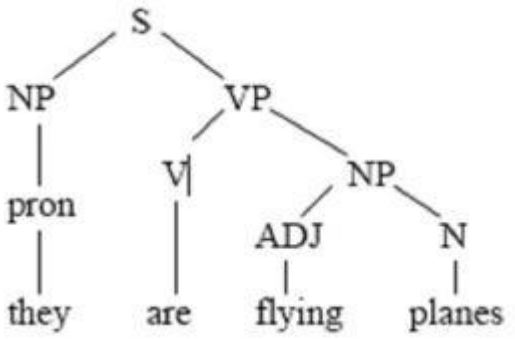
Sebuah kalimat dapat saja mempunyai lebih dari satu pohon.

Contoh 2 :

Diketahui grammar G_2 = {S → SOS A , O → * +, A → 0 1 2 … 9} Kalimat : 2*3+7 mempunyai dua pohon sintaks berikut :



Setelah terbentuk CFG yang telah dinormalkan secara CNF, dalam implementasi parsing, terdapat algoritma yang berguna untuk menentukan apakah suatu untai ‘valid’, atau dapat diciptakan dari aturan-aturan CFG yang ada. Salah satu algoritma yang dapat dipakai adalah Algoritma Cocke-Younger-Kasami (CYK). Algoritma ini menyelesaikan masalah analisa kembali sebuah sub-untai yang sama karena seharusnya analisa sub-untai independen terhadap parsing sub-untai yang diparsing setelahnya. Dengan Program Dinamis, independensi yang diinginkan dapat dicapai ketika parsing. Algoritma CYK termasuk dalam bidang Program Dinamis karena algoritma ini membangun tabel status dua dimensi ketika parsing dimana penentuan parsing selanjutnya diturunkan atau dihasilkan dari parsing sebelumnya, hingga akhir untai. Selain untuk mengetahui validitas untai dalam suatu CFG, algoritma CYK yang dimodifikasi dapat dipergunakan pula untuk membangun pohon parsing.



Pohon parsing yang terbentuk dari sebuah bahasa natural

Push Down Automata (PDA)

PDA adalah mesin otomata dari TBBK yang diimplementasikan dengan stack sehingga hanya terdapat operasi “push” dan “pop” Stack (tumpukan) adalah suatu struktur data yang menggunakan prinsip LIFO (Last In First Out). Sebuah stack selalu memiliki top of stack dan elemen-elemen stack itu yang akan masuk ke dalam stack dengan method “push” dan akan keluar dari stack dengan method “pop”.

- Definisi : PDA adalah pasangan 7 tuple M = (Q, Σ, , q_0 , Z_0 , δ, A), dimana :

Q : himpunan hingga stata, Σ : alfabet input, : alfabet *stack*, $q_0 \in Q$: stata awal, $Z_0 \in$: simbol awal *stack*, A ⊆ Q : himpunan stata penerima,

fungsi transisi δ : Q · (Σ ∪ {ε}) · → $2^{Q \cdot \ *}$ (himpunan bagian dari Q · *)

- Untuk stata q ∈ Q, simbol input a ∈ Σ, dan simbol *stack* X∈ , δ(q, a, X) = (p, α) berarti : PDA bertransisi ke stata p dan mengganti X pada stack dengan string α.
- Konfigurasi PDA pada suatu saat dinyatakan sebagai triple (q, x, α), dimana :

q ∈ Q : stata pada saat tersebut, x ∈ Σ* : bagian string input yang belum dibaca, dan α ∈ * : string yang menyatakan isi *stack* dengan karakter terkiri menyatakan *top of stack*.

- Misalkan (p, ay, Xβ) adalah sebuah konfigurasi, dimana : a ∈ Σ, y ∈ Σ*, X ∈ , dan β ∈ *. Misalkan pula δ(p, a, X) = (q, γ) untuk q ∈ Q dan γ ∈ *. Dapat kita tuliskan

bahwa : (p, ay, Xβ) ⇒ (q, y, γβ).

Sebuah PDA dinyatakan dengan :

Q = himpunan state

Σ = himpunan simbol input

T = simbol stack

S = state awal

F = state akhir

Z = top of stack

PDA memiliki 2 jenis transisi, yaitu yang merima simbol input, simbol top of stack, dan state. Setiap pilihan terdiri dari state berikutnya dan simbol- simbol. Penggantian isi stack dilakukan dengan opersi push dan pop. Jenis transisi yang kedua adalah transisi ε. Transisi ε

tidak melakukan pembacaan input namun hanya menerima simbol top of stack dan state. Transisi ini memungkinkan PDA untuk memanipulasi isi stack dan berpindah antar state tanpa membaca input.

TURING MACHINE

Mesin Turing mengacu pada mesin hipotetis yang diusulkan oleh Alan M. Turing (1912-1954) pada tahun 1936 yang perhitungannya dimaksudkan untuk memberikan definisi operasional dan formal dari gagasan intuitif tentang komputabilitas dalam domain diskrit. Ini adalah perangkat digital dan cukup sederhana untuk menerima analisis teoretis dan cukup kuat untuk merangkul segala sesuatu dalam domain diskrit yang dapat dihitung secara intuitif. Seolah itu tidak cukup, dalam teori komputasi banyak kelas kompleksitas utama dapat dengan mudah dicirikan oleh mesin Turing yang dibatasi dengan tepat; terutama kelas penting P dan NP dan akibatnya pertanyaan utama apakah P sama dengan NP

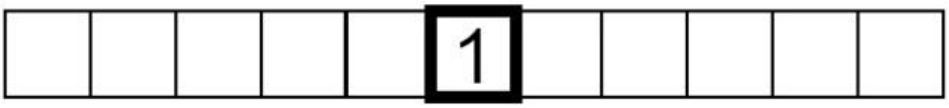
Di atas adalah representasi yang sangat sederhana dari mesin Turing. Ini terdiri dari pita yang sangat panjang yang bertindak seperti memori di komputer biasa, atau bentuk penyimpanan data lainnya. Kotak pada pita biasanya kosong di awal dan dapat ditulis dengan simbol. Dalam hal ini, mesin hanya dapat memproses simbol 0 dan 1 dan " " (kosong), dan dengan demikian dikatakan sebagai mesin Turing 3-simbol.

Pada suatu saat, mesin memiliki kepala yang diposisikan di atas salah satu kotak pada pita. Dengan kepala ini, mesin dapat melakukan tiga operasi yang sangat mendasar:

1. Baca simbol pada kotak di bawah kepala.
2. Edit simbol dengan menulis simbol baru atau menghapusnya.
3. Pindahkan pita ke kiri kanan satu kotak sehingga mesin dapat membaca dan mengedit simbol pada kotak yang berdekatan.

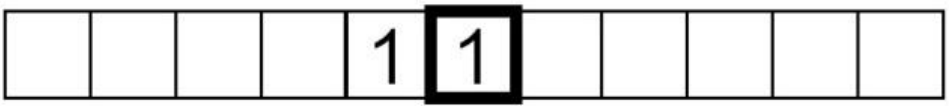
Sebagai contoh sederhana untuk mendemonstrasikan operasi ini, mari kita coba mencetak simbol "1 1 0" pada kaset yang awalnya kosong:

Pertama, kami menulis 1 di kotak di bawah kepala:



Selanjutnya, kami memindahkan kaset ke kiri satu kotak:

Sekarang, tulis 1 di kotak baru di bawah kepala:



Kami kemudian memindahkan rekaman itu ke kiri satu kotak lagi:

Akhirnya, tulis 0 dan hanya itu!