CONTEXT FREE GRAMMAR DAN PUSHDOWN AUTOMATA

OBJEKTIF :

1. Mahasiswa mampu memahami konsep Context Free Grammar
2. Mahasiswa mampu memahami konsep Bentuk Normal CFG
3. Mahasiswa mampu memahami konsep Pushdown Automata

4.1 Context Free Grammar

4.1.1 Mengenal Context Free Grammar

Context Free Grammar adalah bagian dari context free language yang merupakan suatu susunan konteks bahasa untuk pushdown automata. Grammar pada automata adalah suatu susunan variabel dan terminal yang mendefinisikan suatu bahasa. Grammar digambarkan dengan struktur *parse tree*, yaitu suatu struktur grammar dengan penempatan string dari bahasanya. *Parse Tree* adalah produk dari parser pada bahasa pemrograman. Terdapat suatu mesin automaton untuk mendeskripsikan *context free language*, yaitu pushdown automaton.

CFG didefinisikan dengan 4 tuple, yaitu `V, T, P` dan `S` dengan keterangan:

- `V` : himpunan variabel.
- `T` : himpunan terminal.
- `P` : himpunan aturan produksi.
- `S` : Start variabel.

Penjelasan:
Variabel pada grammar adalah representasi dari suatu bahasa dan himpunan string. Pada umumnya, variabel dinotasikan dengan **huruf kapital**. Terminal adalah himpunan simbol yang membentuk string dari bahasa yang didefinisikan. Terminal dinotasikan edngan **huruf kecil** atau simbol tertentu. Aturan produksi pada grammar merupakan suatu daftar aturan untuk setiap variabel yang kana memproduksi terminal-terminal tertentu. Start variabel adalah awal dimulainya suatu grammar pada variabel tertentu.

Contoh:
Terdapat grammar `G = ({S, A}, {x, y}, P, S)` dengan aturan produksi P:
`S → xAy | A | xy | ϵ`
`A → y | xS | yA | ϵ`

Grammar ini akan menghasilkan string `xy, y, xyy, xxxyy, …dst`.

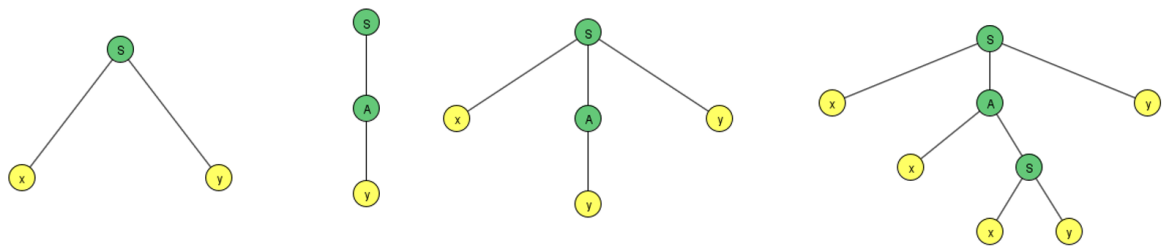
4.1.2 Pohon Derivasi

Pohon derivasi merupakan pohon terurut yang node-nodenya telah diberi label dengan alfabet variabel dan anaknya merupakan hasil dari produksinya.

Contoh:
Terdapat grammar `G = ({S, A}, {x, y}, P, S)` dengan aturan produksi P:
`S → xAy | A | xy | ϵ`
`A → y | xS | yA | ϵ`

Grammar ini akan menghasilkan string `xy, y, xyy, xxxyy`, ...dst. String tersebut dihasilkan dari pohon derivasi berikut:

String xy String y String xyy String xxxyy



**Cara membacanya: String `xy` menghasilkan bentuk pohon derivasi yang pertama(paling kiri), string `y` menghasilkan bentuk pohon derivasi kedua, dan seterusnya.

4.1.3 Bentuk Derivasi

Sebuah pohon derivasi memiliki bentuk derivasi, terdapat 3 bentuk penerjemahan grammar dari aturan produksinya, yaitu:

- Derivasi kiri
- Derivasi kanan
- Derivasi campuran.

Contoh:
Terdapat grammar G dengan aturan produksi:

1. `S → 0SS`
2. `S → 1`

Jawab:
Derivasi Kiri
S → 0**S**S
S → 00**S**SS (produksi 1)
S → 001**S**S (produksi 2, disebut derivasi kiri karena variabel yang diterjemahkan adalah variabel sebelah kiri)
S → 0011**S** (produksi 2)
S → 00111 (produksi 2)
String yang dihasilkan dari derivasi kiri tersebut adalah `00111`

Derivasi Kanan
S → 0**S**S
S → 00**S**SS (produksi 1)
S → 000SSS**S** (produksi 1)
S → 000SS**S**1 (produksi 2, disebut derivasi kanan karena variabel yang diterjemahkan adalah variabel sebelah kanan)
S → 000S**S**11 (produksi 2)
S → 000**S**111 (produksi 2)
S → 0001111 (produksi 2)
String yang dihasilkan dari derivasi kanan adalah 0001111

Derivasi Campuran
S → 0S**S**
S → 0**S**1 (produksi 2, derivasi kanan)
S → 00**S**S1 (produksi 1)
S → 001**S**1 (produksi 2, derivasi kiri)

S → 00111 (produksi 2)
String yang dihasilkan dari derivasi campuran adalah 00111

4.1.4 Rekursif Kiri

Sebuah grammar yang memiliki rekursif kiri harus dieliminasi. Rekursif kiri menyebabkan pohon derivasi terus menerus turun ke kiri dan tidak menghasilkan terminal. Grammar yang memiliki rekurif kiri adalah grammar yang mengandung **minimal 1 α dan 1 β**. Notasi grammar α dan β pada grammar adalah `A → A α | β` (A menghasilkan A alpha dan beta). α adalah semua simbol terminal yang mengikuti variabel pemroduksinya. Sedangkan β adalah simbol terminal hasil produksi lainnya yang tidak mengandung alfabet pemroduksi di sebelah kirinya. Untuk menghilangkan rekurisf kiri, maka perlu dibuat aturan produksi baru yang mengikuti rumus:
`A → β A'`
`A' → α A | ϵ`

Contoh:
Terdapat grammar G dengan aturan produksi `D → Dx | Dy | Dz | a`
Pada grammar tersebut terdapat aturan produksi yang memiliki rekursif kiri, sehingga perlu dihilangkan. Terdapat 3α, yaitu `α1 = x; α2 = y; α3 = z`, sedangkan beta hanya satu, yaitu `β = a.`

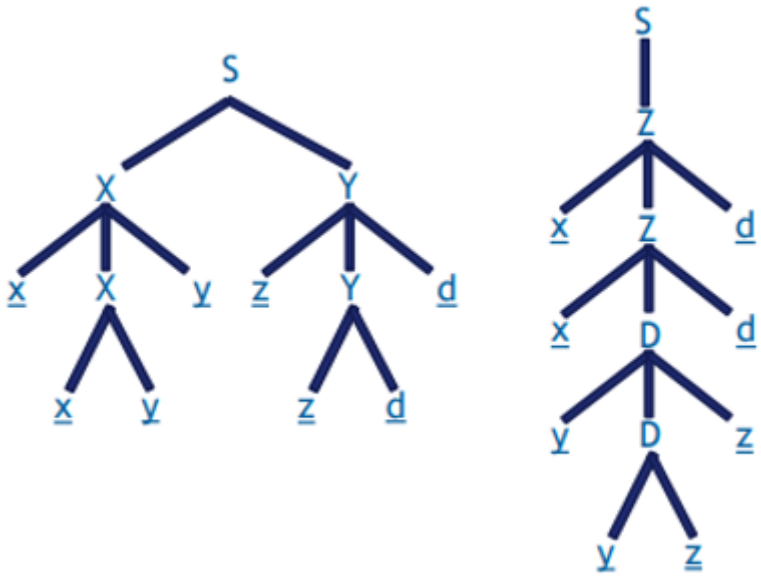
Dengan demikian, dapat dibuat aturan produksi baru sesuai rumus `A → β A'` dan `A' → α A | ϵ` , sehingga menjadi
`D → aD'`
`D' → xD' | yD' | zD' | ϵ`

4.1.5 Grammar Ambigu

Suatu grammar disebut ambigu apabila CFG tersebut memiliki setidaknya satu string dalam bahasa L(G) yang dapat diturunkan dari dua atau lebih pohon derivasi. Ambiguitas suatu grammar tidak mempengaruhi hasil penerjemahan, hanya saja dapat membingungkan *compiler*, karena dihasilkan dari pohon derivasi yang berbeda.

Contoh:
Terdapat CFG G dengan aturan produksi:
`S → XY | Z`
`X → xXy | xy`
`Y → zYd | zd`
`Z → xZd | xDd`
`D → yDz | yz`

String `xxyyzzdd` memiliki dua pohon derivasi sehingga menimbulkan ambiguitas. Dua pohon derivasi tersebut adalah seperti berikut:



4.1.6 Menghilangkan Simbol Tidak Berguna

Suatu simbol X disebut tidak berguna apabila X tidak menurunkan terminal dan X tidak bisa diturunkan dari S. Produksi disebut tidak berguna apabila produksi memuat variabel yang tidak memiliki penurunan yang akan menghasilkan terminal. Produksi tidak akan pernah dapat dicapai dengan penurunan apapun dari simbol awal, sehingga produksi terdeput redundan.

Contoh:
Terdapat grammar dengan aturan produksi:
`S → cSc | Abd | Bdf`
`A → Adc`
`B → cB |`
Hilangkan simbol yang tidak berguna pada grammar tersebut!

Jawab:
Simbol variabel A tidak memiliki penurunan terminal, sehingga bisa dihilangkan. Konsekuensi dari penghilangan produk `A → Adc` maka aturan produksi `S → Abd` tidak memiliki penurunan, sehingga harus dihilangkan juga. Bentuk dari penyederhanaan tersebut akan menghasilkan
`S → cSc | Bdf`
`B → cB | a`

4.2 Bentuk Normal CFG

Suatu CFG memiliki tujuan untuk membentuk suatu aturan produksi yang normal. Terdapat dua bentuk normal dari CFG, yaitu bentuk normal Chomsky (Chomsky Normal Form / CNF) dan bentuk normal Greibach (Greibach Normal Form / GNF).

4.2.1 Penghilangan Unit Produksi ε

Untuk menghilangkan unit produksi ε (epsilon), hal pertama yang perlu dilakukan adalah menentukan simbol yang *"nullable"*, yaitu simbol yang menghasilkan produksi ε.

Contoh:
Terdapat grammar :
`S → PQ`
`P → xPP| ϵ`
`Q → yQQ | ϵ`

Pada grammar tersebut, simbol yang memiliki *nullable* adalah P dan Q, karena menghasilkan ε. Apabila P dan Q adalah *nullable*, maka S yang menghasilkan PQ dapat juga disebut *nullable*. Produksi akhir yang kemungkinan didapat dari S adalah S → PQ | P | Q. Hasil P didapat apabila S → PQ, Q nya akan menghasilkan ε. Hasil Q didapat jika S → PQ, P nya menghasilkan ε. Produksi akhir yang didapat dari S adalah `S → PQ | P | Q`.

Untuk produksi `P → xPP | ϵ`, dapat menghasilkan produksi `P → xPP | xP | xP | x`. Terdapat dua hasil produksi xP, hal itu disebabkan oleh ε yang menggantikan P pada xPP, jika P yang pertama digantikan ε akan menghasilkan xP, dan jiika P yang kedua digantikan ε, maka akan menghasilkan xP juga. Sedangkan untuk hasil x, didapat dari xPP dengan kedua simbol P diganti ε. Karena hasil xP sama, maka dapat diambil salah satu saja sehingga simbol produksi P menjadi `P → xPP | xP | ϵ`.

Untuk produksi `Q → yQQ | ϵ`, dapat menghasilkan `Q → yQQ | yQ | yQ | q`. Konsep padad produksi Q sama dengan produksi P. Sehingga akan menghasilkan simbol `Q → yQQ | yQ | q`.

Sehingga bentuk aturan produksi pada grammar yang sudah dihilangkan unit produksi ε nya menjadi:
`S → PQ | P | Q`
`P → xPP | xP | ϵ`
`Q → yQQ | yQ | q`

4.2.2 Chomsky Normal Form

Chomsky Normal Form memiliki aturan produksi tanpa ε dan berbentuk `A → BC` atau `A → a`, dengan `A, B ϵ Vn dan a ϵ Tn`. CNF dapat dibentuk dengan algoritma:

1. Eliminasi unit produksi yang tidak berguna dan produksi ε jika ada.
2. Eliminasi terminal di sisi kanan dengan panjang dua string atau lebih.
3. Batasi jumlah variabel di sisi kanan produksi kedua.

Contoh:
Ubahlah aturan prouksi berikut ke dalam bentuk CNF!
`S → xAyB`
`A → Ca | a`
`B → bB | b`
`C → c`
Terdapat dua aturan produksi yang sudah CNF, yaitu `A → a` dan `B → b`, untuk S dan C belum dalam bentuk CNF.

Produksi Awal	Produksi Baru	CNF yang dihasilkan
`S → xAyB`	`S →XAYB, X→x, Y→y`	`X → x, Y → y`
`S → XAYB`	`S → XaYb, Xa = XA, Yb = YB`	`S → XaYb, Xa = XA, Yb = YB`
`A → Aa`	`A → CaA, Ca → a`	`A → CaA, Ca → a`
`B → bB`	`B → CbB, Cb → b`	`B → CbB, Cb → b`

Setelah dilakukan perubahan, maka terbentuk aturan produksi yang sudah dalam bentuk CNF, yaitu:

- `S → XaYb`
- `X → x`
- `Y → y`

- `Xa = XA`
- `Yb = YB`
- `A → CaA | a`
- `B → CbB | B`
- `Ca → a`
- `Cb → b`

4.2.2 Greibach Normal Form

Greibach Normal Form adalah grammar yang digunakan pada mesin automaton yang disebut pushdown automata. GNF diperkenalkan oleh Sheila Greibach. Suatu produksi grammar disebut sudah berbentuk GNF apabila aturan produksi berbentuk `A → ax`. dengan `a ∈ T` dan `x ∈ V*`. Pada grammar GNF, setiap produksi memiliki satu simbol terminal saja atau satu simbol terminal diikuti oleh rangkaian variabel. Untuk dapat membuat GNF, setiap grammar **harus sudah berbentuk CNF** dan tidak memiliki simbol yang tidak berguna, serta tidak memiliki produksi ε. Mesin *pushdown automata* membaca aturan produksi grammar yang sudah memenuhi GNF untuk menentukan suatu string diterima atau ditolah oleh pushdown automata.

Terdapat algoritma untuk membentuk GNF, yaitu:

1. Lemma G1:
   Misalkan `G = (V, T, P, S)` dari suatu CFG.
   `A → α1βα2` suatu produksi dalam P
   `B → β1 | β2 |…| βn` himpunan produksi B

   Misalkan `G1 = (V, T, P, S)` diperoleh dengan menghilangkan semua produksi dalam bentuk:
   `A → α1βα2` dari P dan menambahkan produksi dalam bentuk:
   `A → α1β1α2 | α1β2α2 | … | α1βnα2`
   Maka `L(G) = L(G1)`

2. Lemma G2:

   Lemma G2: Misalkan `G = (V, T, P, S)` dari suatu CFG
   `A → Aα1 | Aα2 | … | A*α*n` adalah himpunan sebagian produksi A, sedangkan produksi lainnya dalam bentuk:
   `A → β1 | β2 |… | βs*`

   Misalkan `G1 = (V ∪ {B}, T, P1, S)` dibentuk dengan menambahkan variabel B ke V dan mengganti semua produksi A dengan produksi dalam bentuk

   $$\begin{matrix} A \to \beta i \\ A \to \beta i B \end{matrix} \Big\} \; 1 \le i \le s$$

   $$\begin{matrix} B \to \alpha i \\ B \to \alpha i B \end{matrix} \Big\} \; 1 \le i \le n$$

   Maka `L(G1) = L(G)`

Contoh 1:
Berikut adalah contoh bentuk aturan produksi yang sudah GNF, yaitu sebagai berikut:
`S → a | aAB`
`A → aBCD`
`B → b`
`C → cB`
`D → d | aBC`

Contoh 2:
Konversikan aturan produksi berikut menjadi GNF!
`S → CA`
`A → a | b`
`B → b | AC`
`C → BD | c`
`D → AB | d`

Masing-masing variabel diberikan urutan simbol variabel, yaitu `S, A ,B, C, D`. Dari aturan tersebut terdapat aturan yang tidak sesuai urutannya, yaitu `C → BD | c` (C menghasilkan BD dan C), dan `D → AB | d` (D menghasilkan AB dan d).

Aturan yang perlu diubah terlebih dahulu dengan urutan sebagai berikut:
(Untuk aturan produksi A tidak dilakukan konversi karena sudah memenuhi persyaratan GNF)

1. `D → AB | d`; Nilai A pada aturan D ini, digantikan dengan aturan produksi A yang sudah ditentukan yaitu a dan b. Sehingga nilai D berubah menjadi:
   `D → aB | bB | d`; aturan produksi ini sudah memenuhi GNF
2. `C → BD | c`; Nilai B digantikan dengan aturan produksi yang sudah ditentukan yaitu b dab AC. Sehingga menjadi:
   `C → bD | ACBD | c`; pada aturan produksi ini masih terdapat aturan yang tidak sesuai dengan urutannya, yaitu `C → ACBD`, sehingga A digantikan dengan `a | b`.
3. `C → bD | aCBD | bCBD | c`; Aturan produksi ini sudah diubah pada nilai A nya dan sudah memenuhi GNF.
4. Aturan `B → b | AC`; Nilai A digantikan dengan aturan produksi yang sudah ditentukan yaitu a dan b. Sehingga menjadi: `B → b | aC | bC` .
5. Pada aturan `S → CA`. nilai C digantikan dengan C yang sudah GNF (langkah 3) sehingga aturan produksinya menjadi `S → bDA | aCBDA | bCBDA | cA`.

Sehingga aturan produksi **yang sudah memenuhi GNF** adalah:
`S → bDA | aCBDA | bCBDA | cA`
`A → a | b`
`B → b | aC | bC`
`C → bD | aCBD | bCBD | c`
`D → aB | bB | d`

4.3 Pushdown Automata

Pushdown Automata didefinisikan oleh context free language. Pushdown Automata merupakan perpanjangan dari NFA dengan penambahan suatu tumpukan (*stack*) yang sama dengan prinsip tumpukan pada struktur data. Tumpukan ini dapat dibaca, ditambah (*push*), dan dibuang (*pop*). Pada penggunaannya, pushdown automata dapat menerima string dengan berbagai cara, seperti:

- Pushdown automata menerima string apabila berhenti di final state.
- Pushown automata menerima string apabila semua input dan tumpukannya sudah habis, tidak terpengaruh dengan input berhenti di final state atau tidak.

Pushown Automata memiliki cara kerja yang sama dengan finite automata, yaitu dengan membaca input an bergerak sesuai dengan fungsi transisi yang dimiliki automaton tersebut. Setiap perpindahan fungsi transisi juga melibatkan tumpukan, setiap fungsi transisi dapat menambah tumpukan, membuang tumpukan, atau hanya membaca saja tanpa ditambah dan dibuang.

Pushown Automata didefinisikan dengan 7 tuple `M = (Q, Σ, Γ, δ, q0, Z, F)`, dengan keterangan:

`Q` : himpunan state jumlah terbatas

`Σ` : alfabet input (Simbol Sigma)

`Γ` : alfabet tumpulan (Simbol Gamma)

`δ` : fungsi transisi (Simbol Delta)

`q0` : start state

`Z` : start stack

`F` : final state

4.3.1 Menggambar Pushown Automata

Biasanya pushown automata tidak dapat digambar, namun engan beberapa penambahan input, pushdown automata dapat digambar menggunakan JFLAP.

Contoh:

Terdapat pushdown automata dengan 7-tuple `({q0, q1, q2}, {0, 1}, {B, M}, δ, q0, B, q2)` dengan fungsi transisi:

`δ(q0, 0 , B) = (q1, MB)`

`δ(q0, ϵ , B) = (q2, ϵ)`

`δ(q1, 0 , M) = (q1, MM)`

`δ(q1, 1, M) = (q2, ϵ)`

`δ(q1, 1 , B) = (q2, ϵ)`

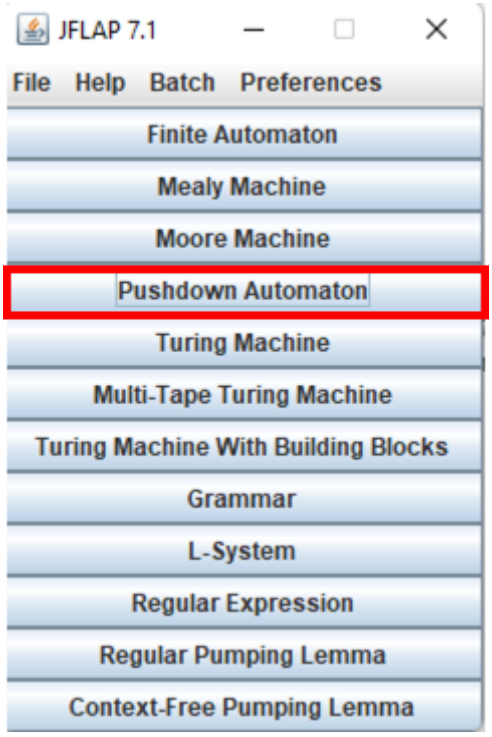
`δ(q2, ϵ , B) = (q2, ϵ)`

`δ(q2, 0 , B) = (q2, ϵ)`

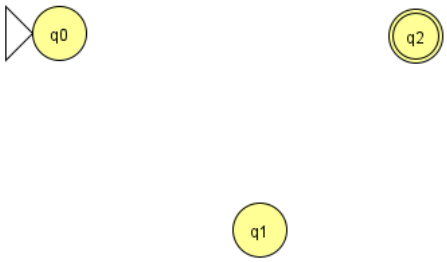
`δ(q2, 1 , B) = (q2, ϵ)`

Langkah 1:

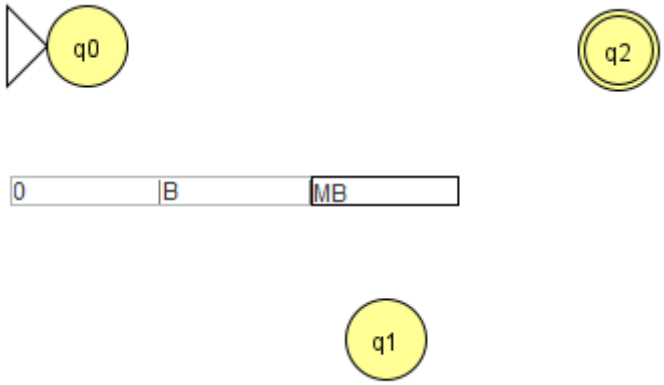
Buka aplikasi JFLAP, kemudian pilih Pushdown Automata , lalu pilih *Multiple Char....*, Klik OK



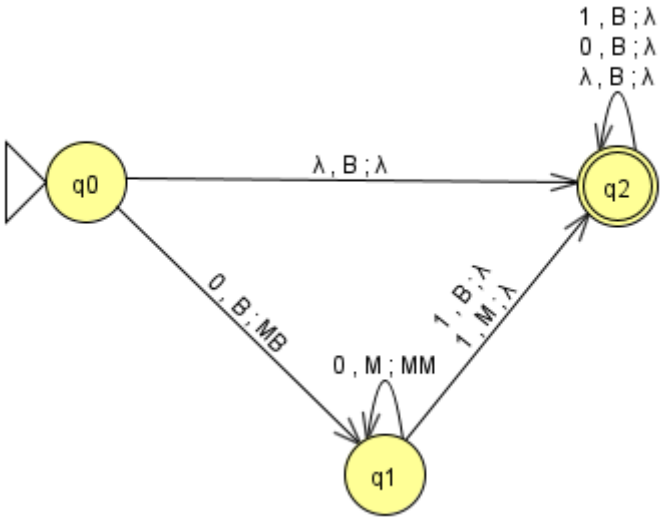
Langkah 2:
Buat 3 state seperti berikut



Langkah 3:
Berikan transisi menggunakan *Transition Creator*, masukkan aturan transisi yang sudah ditentukan.



Sambungkan semua transisi sehingga seperti gambar berikut.



Langkah 4:
Lakukan pengujian dengan menggunakan menu `Input → Fast Run → Masukkan Input → Pilih Empty Stack/Final State`, sebagai contoh:

- Input `01` dengan empty stack menampilkan hasil *rejected* atau ditolak.
- Input `01` dengan final state menampilkan hasil *rejected*.
- Input `011` dengan empty stack menampilkan hasil *rejected.*
- Input `011` dengan final state menampilkan hasil *rejected*.

4.3.2 Eksekusi Pushown Automata

Pushdown Automata dapat dieksekusi dengan udah melalui pelacakan fungsi transisi dan konsumsi input. Apabila input dan tumpukan sudah habis, maka input tersebut diterima.

Contoh:
Terdapat Pushdown Automata A dengan 7 Tuple = `({q0, q1, q2}, {a, b}, {B, M}, δ, q0, B, q2)`.
Tentukan apakah string aabb dan abba diterima dengan fungsi transisi berikut:
`δ(q0, a , B) = (q1, MB)`
`δ(q0, b , B) = (q1, BB)`
`δ(q0, ϵ, B) = (q2, ϵ)`
`δ(q1, a, M) = (q1, MM)`
`δ(q1, b, M) = (q2, ϵ)`
`δ(q2, b, M) = (q2, ϵ)`
`δ(q2, b, B) = (q2, ϵ)`
`δ(q2, ϵ, B) = (q2, ϵ)`

Catatan: Pada δ(q0, a , B), q0 merupakan state, a merupakan input, dan q0 adalah tumpukan.

Solusi:
Pushdown Automata menerima input string apabila input dan stack habis. Penyelesaian input string `aabb` dan abba dilakukan dengan melacak pergerakan transisi masing-masing state dan stack.

Input aabb, dimulai dari transisi pertama yaitu δ(q0, a , B).
Penjelasan:
Jika diperhatikan input aabb diawali dengan huruf a. kemudian perhatikan transisi yang ada pada soal. Kita akan menggunakan transisi awal sesuai dengan input, yaitu `δ(q0, a, B)` dan akan akan melakukan transisi ke `(q1, MB)`.
Setelah dilakukan transisi, maka bentuknya menjadi seperti berikut:

`δ(q0, a , B)` `⊢ (q1, abb, MB)` (String pertama dihilangkan, kemudian tambahkan M)

Gunakan transisi sebelumnya yaitu (q1, abb, MB) kemudian cari transisi yang sesuai dengan aturan transisi yang menggunakan q1, berawalan a, dan M (Untuk B tidak dibaca). Transisi yang didapat sesuai aturan itu adalah `δ(q1, a, M) = (q1, MM)` , Setelah dilakukan transisi maka bentuknya menjadi seperti berikut: `(q1, bb, MMB)` (String a dihilangkan, kemudian tambahkan M)

Transisi ini didapat dari transisi `δ(q1, a, M)` sebelumnya, yaitu q1 dengan string a)

Setelah semuanya ditransisikan, berikut adalah hasil dari apakah string aabb diterima atau tidak

`δ(q0, aabb, B) ⊢ (q1, abb, MB)`

`⊢ (q1, bb, MMB)`

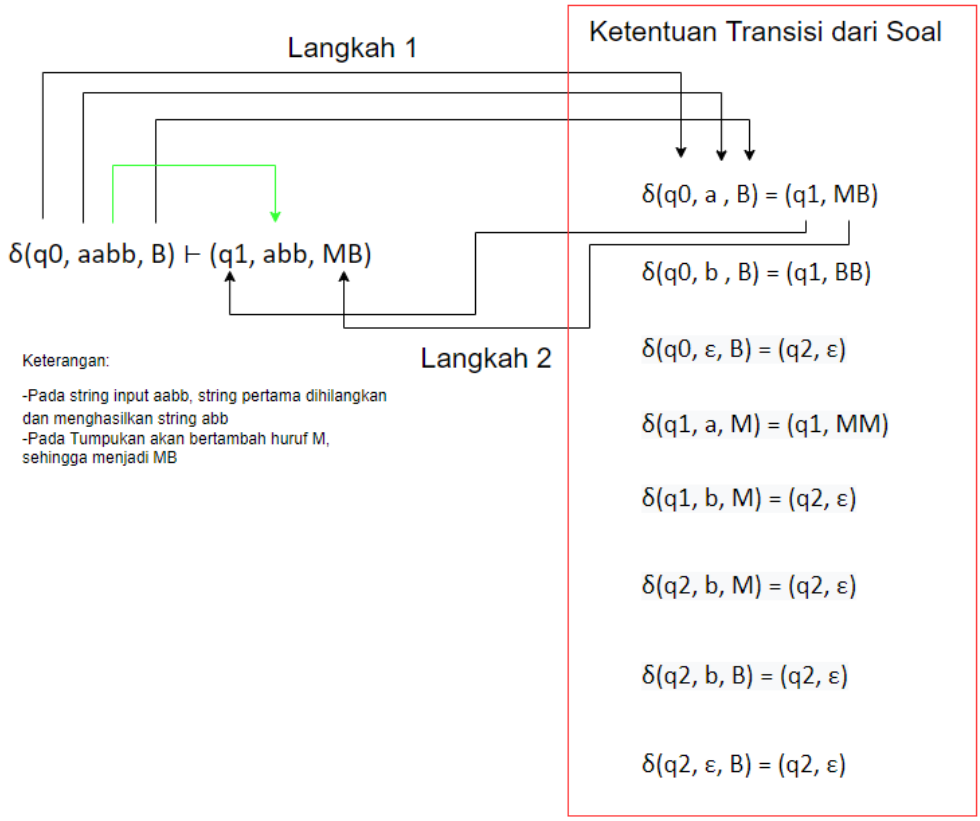
`⊢ (q2, b, MB)`

`⊢ (q2, ϵ, B)`

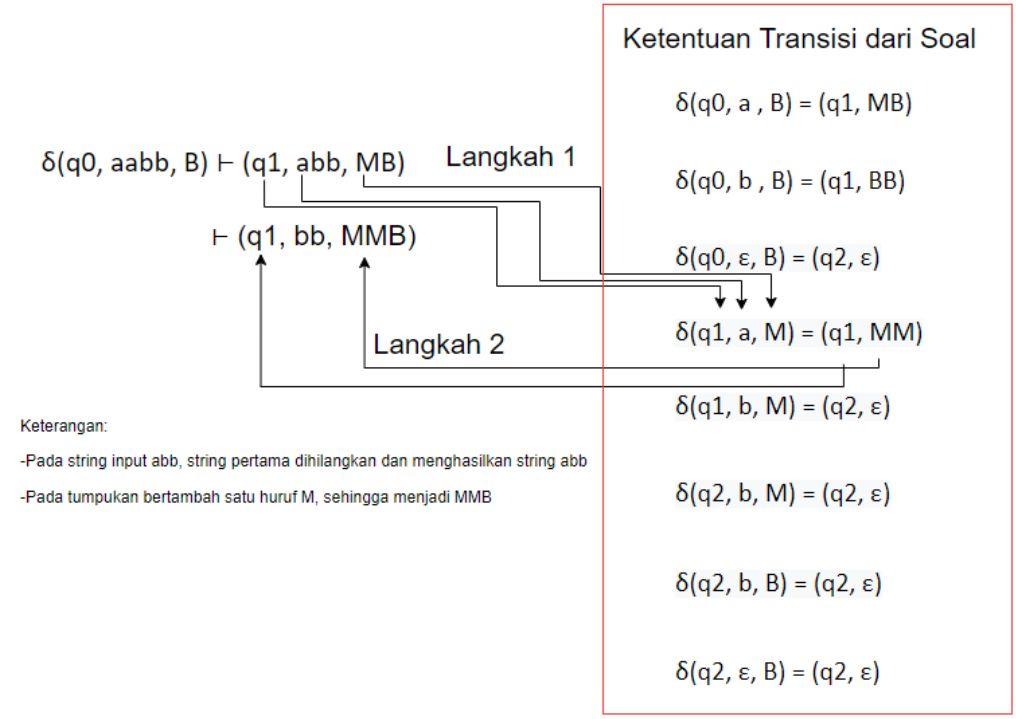
`⊢ (q2, ϵ, ϵ)`

Dapat diambil kesimpulan, input `aabb` **diterima** karena input dan tumpukan sudah habis atau tidak ada sisa.
Berikut adalah tahapan cara penyelesaiannya:

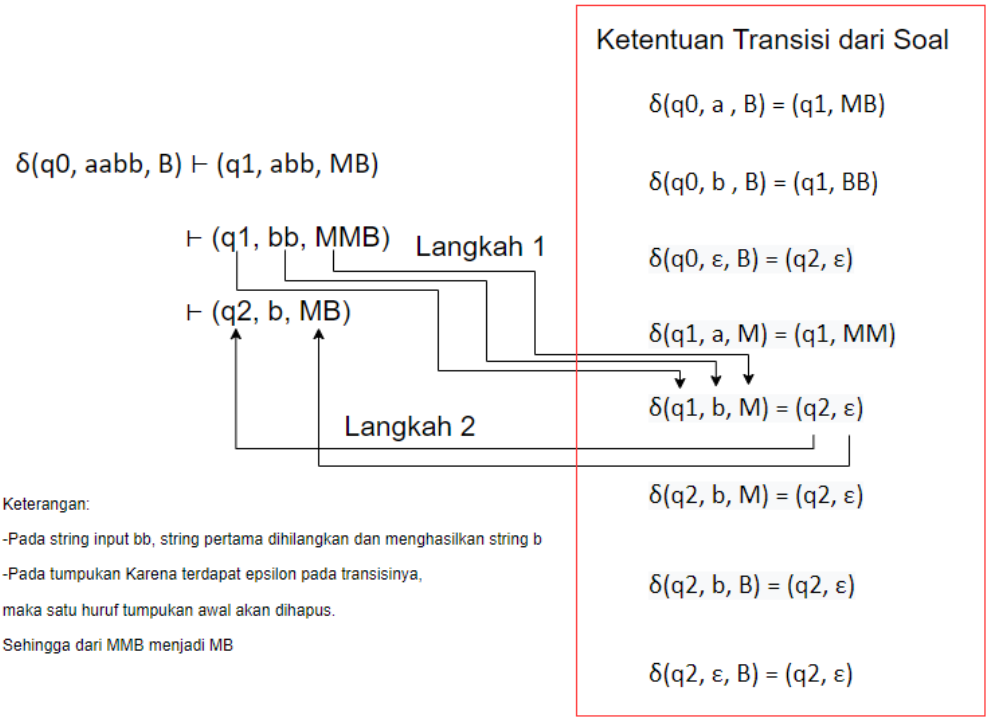
Step 1:


Langkah 1
Ketentuan Transisi dari Soal
δ(q0, a , B) = (q1, MB)
δ(q0, aabb, B) ⊢ (q1, abb, MB)
δ(q0, b , B) = (q1, BB)
δ(q0, ε, B) = (q2, ε)
Langkah 2
Keterangan:
-Pada string input aabb, string pertama dihilangkan dan menghasilkan string abb
-Pada Tumpukan akan bertambah huruf M, sehingga menjadi MB
δ(q1, a, M) = (q1, MM)
δ(q1, b, M) = (q2, ε)
δ(q2, b, M) = (q2, ε)
δ(q2, b, B) = (q2, ε)
δ(q2, ε, B) = (q2, ε)

Step 2:


Ketentuan Transisi dari Soal
δ(q0, a , B) = (q1, MB)
δ(q0, aabb, B) ⊢ (q1, abb, MB)
Langkah 1
δ(q0, b , B) = (q1, BB)
⊢ (q1, bb, MMB)
δ(q0, ε, B) = (q2, ε)
Langkah 2
δ(q1, a, M) = (q1, MM)
δ(q1, b, M) = (q2, ε)
Keterangan:
-Pada string input abb, string pertama dihilangkan dan menghasilkan string abb
-Pada tumpukan bertambah satu huruf M, sehingga menjadi MMB
δ(q2, b, M) = (q2, ε)
δ(q2, b, B) = (q2, ε)
δ(q2, ε, B) = (q2, ε)

Step 3:



Keterangan:

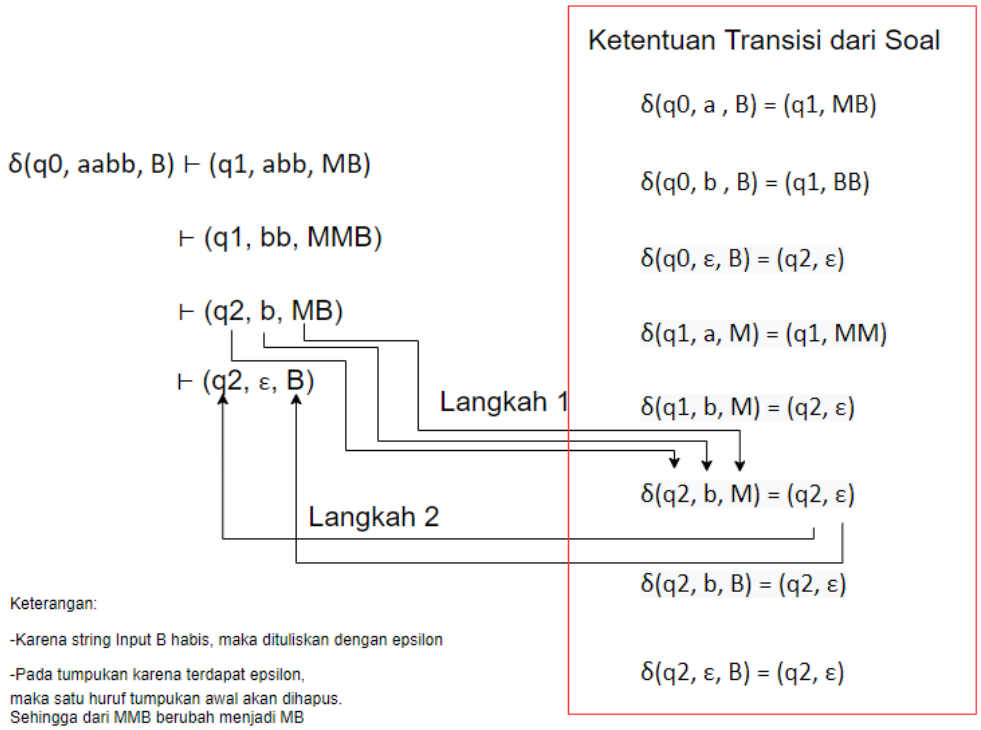
-Pada string input bb, string pertama dihilangkan dan menghasilkan string b

-Pada tumpukan Karena terdapat epsilon pada transisinya,

maka satu huruf tumpukan awal akan dihapus.

Sehingga dari MMB menjadi MB

Step 4:

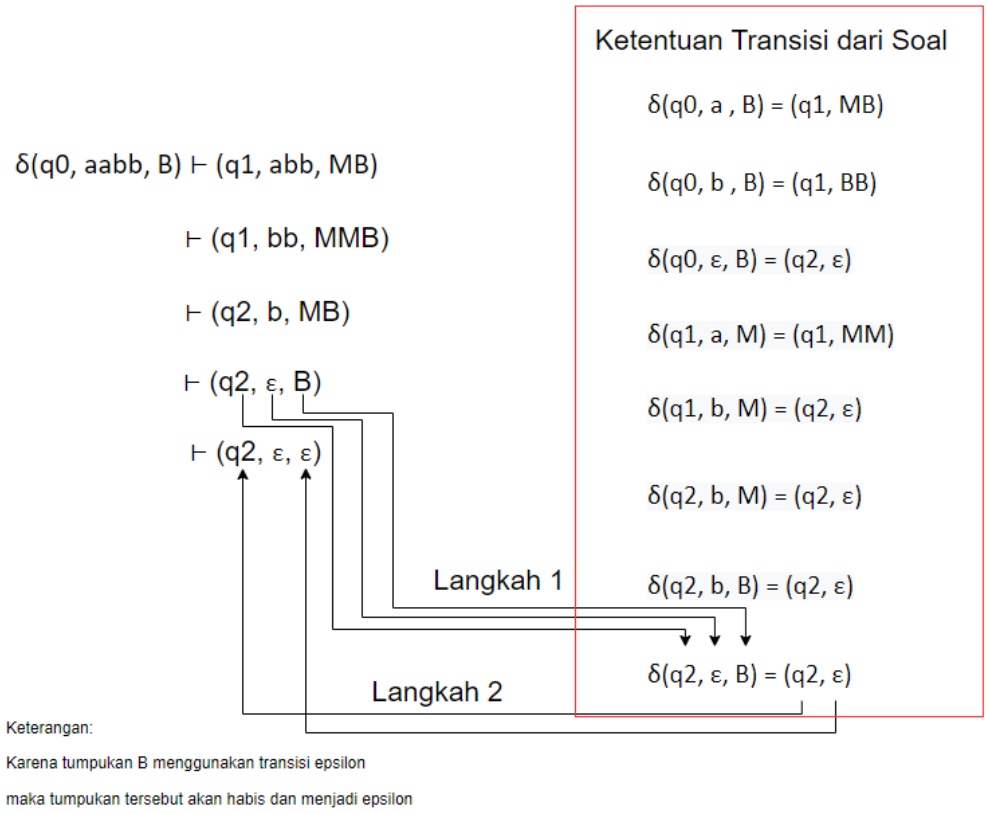


Keterangan:

-Karena string Input B habis, maka dituliskan dengan epsilon

-Pada tumpukan karena terdapat epsilon,
maka satu huruf tumpukan awal akan dihapus.
Sehingga dari MMB berubah menjadi MB

Step 5:



Input aabb diterima karena tumpukan dan input sudah habis

Input aabb

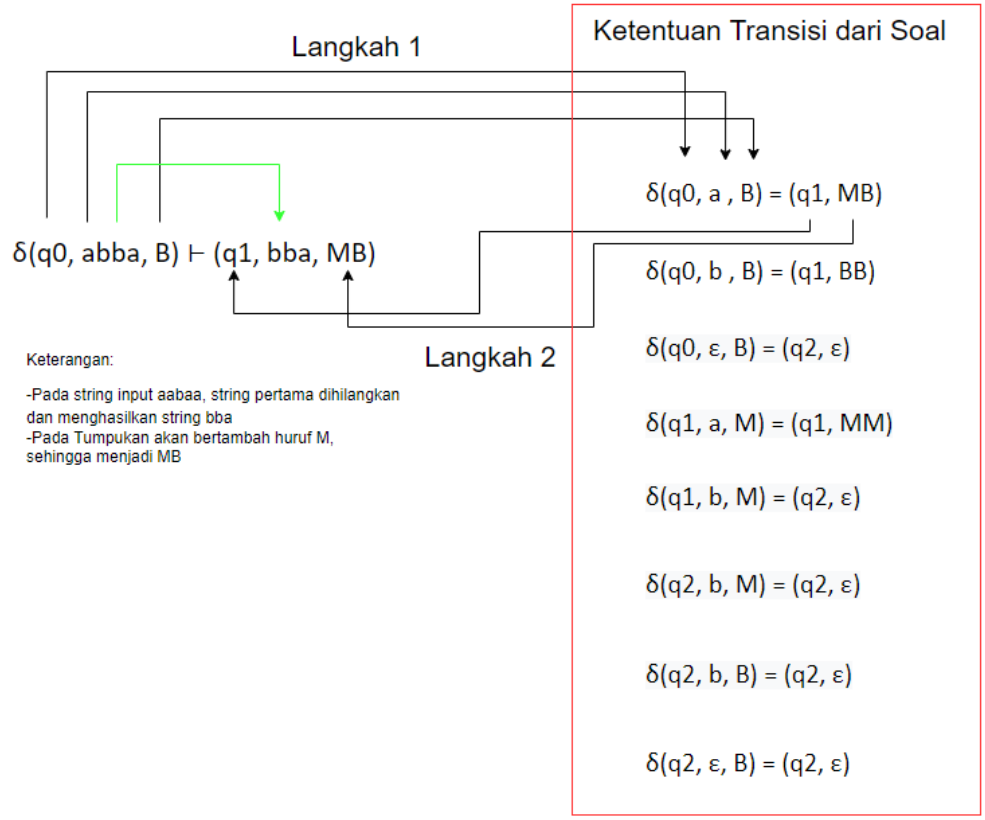
`δ(q0, abba, B) ⊢ (q1, bba, MB)`

`⊢ (q2, ba, B)`

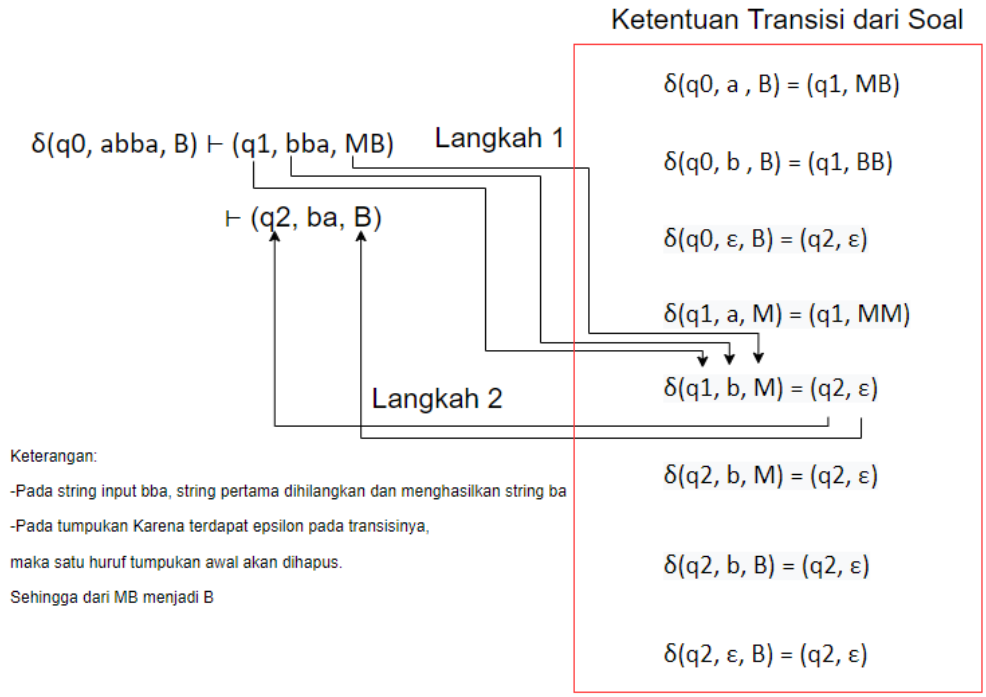
`⊢ (q2, a, ϵ)`

Dapat diambil kesimpulan, input `abba` **ditolak** karena tumpukan sudah habis sementara input masih ada sisa.
Berikut adalah tahapan cara penyelesaiannya:

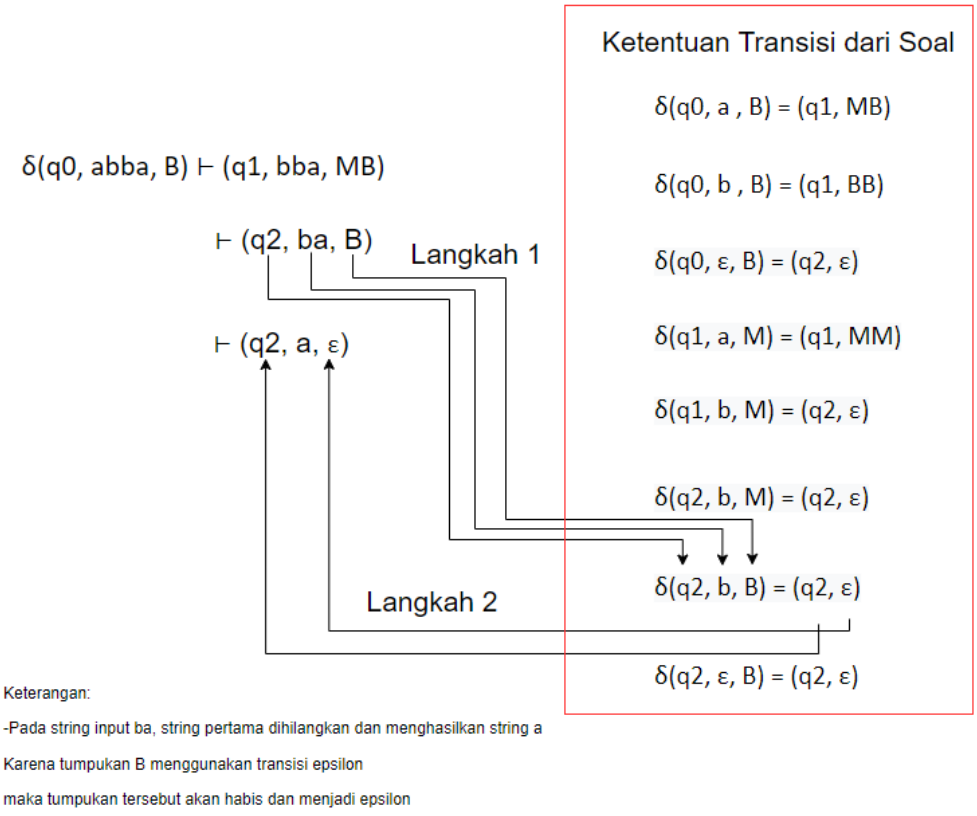
Step 1:


Ketentuan Transisi dari Soal
Langkah 1
δ(q0, a , B) = (q1, MB)
δ(q0, b , B) = (q1, BB)
δ(q0, ε, B) = (q2, ε)
δ(q1, a, M) = (q1, MM)
δ(q1, b, M) = (q2, ε)
δ(q2, b, M) = (q2, ε)
δ(q2, b, B) = (q2, ε)
δ(q2, ε, B) = (q2, ε)
δ(q0, abba, B) ⊢ (q1, bba, MB)
Langkah 2
Keterangan:
-Pada string input aabaa, string pertama dihilangkan
dan menghasilkan string bba
-Pada Tumpukan akan bertambah huruf M,
sehingga menjadi MB

Step 2:


Ketentuan Transisi dari Soal
δ(q0, a , B) = (q1, MB)
δ(q0, b , B) = (q1, BB)
δ(q0, ε, B) = (q2, ε)
δ(q1, a, M) = (q1, MM)
δ(q1, b, M) = (q2, ε)
δ(q2, b, M) = (q2, ε)
δ(q2, b, B) = (q2, ε)
δ(q2, ε, B) = (q2, ε)
δ(q0, abba, B) ⊢ (q1, bba, MB)
Langkah 1
⊢ (q2, ba, B)
Langkah 2
Keterangan:
-Pada string input bba, string pertama dihilangkan dan menghasilkan string ba
-Pada tumpukan Karena terdapat epsilon pada transisinya,
maka satu huruf tumpukan awal akan dihapus.
Sehingga dari MB menjadi B

Step 3:



Input abba ditolak karena tumpukan sudah habis sementara input masih ada

4.3.3 Konversi Context Free Grammar ke Pushdown Automata

Sebuah context free grammar yang sudah berbentuk GNF, dapat dikonversi menjadi pushdown automata sehingga dapat digunakan untuk memproses input string.

Algoritma konversi CFG ke PDA:

1. Inisialisasi pada state q0. State q0 digunakan untuk mentransisikan dari state q0 ke q1 untuk meletakkan start simbol ke dalam tumpukan.
   `δ(q0, ϵ, z) → (q1, Sz)`
2. State q1 melakukan banyak transisi. Setiap transisi akan selalu menuju q1.
3. Perpindahan tttransisi dari q1 ke qf digunakan untuk menerima string.
   `δ(q1, ϵ, z) → (qf, z)`

Contoh:
Konversikan CFG dengan aturan produksi:

`S → b | cXYZ`

`Y → c`

`X → aS`

`Z → bXY`

Tentukan apakah string `baacd` diterima bila tumpukan dimulai dari S!

Solusi:

START	`δ(q0, ϵ, z) → (q1, Sz)`
`S → b`	`δ(q1, a, S) → (q1, ϵ)`
`S → cXYZ`	`δ(q1, c, S) → (q1, XYZ)`
`X → aS`	`δ(q1, a, X) → (q1, S)`
`Y → c`	`δ(q1, c, Y) → (q1, ϵ)`
`Z → bXY`	`δ(q1, b, Z) → (q1, XY)`
FINISH	`δ(q1, ϵ, z) → (qf, z)`

Penjelasan:

- Untuk *Start* sama dengan aturan 1 pada algoritma CFG ke PDA, yaitu `δ(q0, ϵ, z) → (q1, Sz)`.
- Aturan produksi `S → b` setelah dikonversi akan menjadi δ(q1, a, S) → (q1, ε).
- Aturan produksi `S → cXYZ` setelah dikonversi menjadi `δ(q1, c, S) → (q1, XYZ)`. Dimana S menjadi tumpukan, c menjadi input, dan XYZ menjadi tumpukan pada transisi tujuan.
- Aturan produksi `X → aS` setelah dikonversi menjadi `δ(q1, a, X) → (q1, S)`. Dimana X menjadi tumpukan, a menjadi input, dan S menjadi tumpukan pada transisi tujuan.
- Aturan produksi `Y → c` setelah dikonversi menjadi `δ(q1, c, Y) → (q1, ϵ)`. Dimana Y menjadi tumpukan, c menjadi input, dan karena tidak tumpukan pada transisi tujuan (huruf dengan kapital) maka diisi epsilon.
- Aturan produksi `Z → bXY` setelah dikonversi menjadi `δ(q1, b, Z) → (q1, XY)`. Dimana Z menjadi tumpukan, b menjadi input, dan XY menjadi tumpukan pada transisi tujuan.
- Pada *Finish*, karena sudah tidak ada aturan transisi yang tersisa, maka dituliskan dengan `δ(q1, ϵ, z) → (qf, z)` sesuai dengan aturan 3 algoritma konversi CFG ke PDA.

Penentuan string baacd diterima attau ditolak ditelusuri dengan pelacakan:

```
δ(q0, baacd, z) ⊢ (q1, baacd, Sz)
                ⊢ (q1, aacd, z)
```

Karena tidak ada pergerakan, maka input string `baacd` **ditolak** oleh PDA.

4.4 RANGKUMAN

- CFG didefinisikan dengan 4 tuple, yaitu `V, T, P` dan `S`.
- String yang dihasilkan dari suatu CFG dapat diturunkan menggunakan pohon derivasi yang ditelusuri menggunakan aturan produksi CFG tersebut.
- Suatu grammar disebut ambigu apabila CFG tersebut memiliki setidaknya satu string dalam bahasa L(G) yang dapat diturunkan dari dua atau lebih pohon derivasi.
- Produksi disebut tidak berguna apabila produksi tersebut memuat variabel yang tidak memiliki penurunan yang akan menghasilkan terminal dan produksi tidak akan pernah dapat dicapai dengan penurunan apapun dari simbol awal, sehingga produksi tersebut redundan.
- Terdapat dua bentuk normal yang umum dipelajari dalam grammar, yaitu Chomsky Normal Form (CNF) dan Greibach Normal Form (GNF).
- Grammar CNF harus memenuhi syarat variabel yangb menghasilkan 2 variabel atau 1 terminal, tidak memiliki produksi ε, dan tidak memiliki *useless symbol*.
- Grammar GNF harus memenuhi syarat, sebelum dikonversi harus sudah CNF, variabel menghasilkan 1 terminal diikuti lebih dari 0 variabel, tidak memiliki produksi ε, dan tidak memiliki *useless symbol.*
- Grammar GNF digunakan untuk pembuatan mesin automaton, yaitu pushdown automata.
- Pushdown automata digunakan untuk menentukan string ditolak atau diterima, konsepnya mirip seperti finite automata, namun memiliki tumpukan.
- Pushdown automata dapat dihasilkan dari context free grammar yang sudah berbentuk GNF.
- Pushown Automata didefinisikan dengan 7 tuple `M = (Q, Σ, Γ, δ, q0, Z, F)`.
- Context free grammar dapat dikonversi menjadi pushdown automata.

REFERENSI

[1] Harya Bima Dirgantara. 2016. Teori Bahasa dan Automata: Merancang Automaton Dengan JFLAP. Jakarta: Penerbit Teknosain.

[2] John E. Hopcroft, Rajeev Motwani, Jeffrey D. Ullman. 2007. Teori Bahasa dan Otomata Edisi Kedua. Yogyakarta: Penerbit Andi.