Teori Bahasa dan Automata

Push Down Automata dan Parsing

CFG to PDA (1)

- Jika *grammar* G adalah *context free*, kita dapat membangun PDA *non-deterministic* setara yang menerima *language* yang dihasilkan oleh CFG G.

CFG to PDA (2)

- Jika P adalah *pushdown automata*, *grammar* G yang setara *context free* dapat didefinisikan :
  - L (G) = L (P)
- Dimungkinkan konversi dari PDA ke CFG dan sebaliknya.

CFG to PDA (3)

- Input
  - CFG, G = (V, T, P, S)
- Output
  - Equivalent PDA, P = (Q, ∑, S, δ, q0, I, F)

CFG to PDA (4)

- Ubah produksi CFG menjadi GNF *(Greibach Normal Form)*.
- PDA hanya akan memiliki satu status {q}.
- Simbol awal CFG akan menjadi simbol awal di PDA.
- Semua non-terminal CFG akan menjadi simbol stack PDA dan semua terminal CFG akan menjadi simbol input PDA.
- Untuk setiap produksi dalam bentuk $A \rightarrow aX$, di mana a adalah terminal dan A, X adalah kombinasi terminal dan non-terminal, buat transisi $\delta (q, a, A)$.

Contoh 1

- Buat PDA dari CFG berikut.
  - G = ({S, X}, {a, b}, P, S)
- dimana produksinya:
  - $S \rightarrow XS \mid \varepsilon$, $A \rightarrow aXb \mid Ab \mid ab$

Solusi 1

- P = ({q}, {a, b}, {a, b, X, S}, δ, q, S)
- dimana δ –
  - δ(q, ε , S) = {(q, XS), (q, ε)}
  - δ(q, ε , X) = {(q, aXb), (q, Xb), (q, ab)}
  - δ(q, a, a) = {(q, ε)}
  - δ(q, 1, 1) = {(q, ε)}

Parsing (1)

- Parsing digunakan untuk mendapatkan string menggunakan aturan produksi Grammar.
- Digunakan untuk memeriksa penerimaan string.
- Compiler digunakan untuk memeriksa suatu string benar secara sintaksis atau tidak.
- Parser mengambil input dan membangun pohon parse.

Parsing (2) – Jenis Parsing

- Top-Down
  - dimulai dari atas dengan simbol awal dan mendapatkan string menggunakan pohon parse.
- Bottom-Up
  - dimulai dari bawah dengan string dan sampai ke simbol awal menggunakan pohon parse.

Parsing (3) – Langkah Top Down

- Pop *non-terminal* di sisi kiri produksi di bagian atas tumpukan dan *push* string sisi kanannya.
- Jika simbol atas tumpukan cocok dengan simbol input yang sedang dibaca, lepaskan.
- *Push* simbol awal 'S' ke dalam tumpukan.
- Jika string input dibaca sepenuhnya dan tumpukan kosong, lanjutkan ke status akhir 'F'.

Contoh 2

- Desain parser top-down untuk ekspresi
  - x + y * z
  - P: S → S + X | X
  - X → X * Y | Y
  - Y → (S) | I

Solusi 2

- (x+y*z, I) ⊢
- (x+y*z, SI) ⊢
- (x+y*z, S + XI) ⊢
- (x+y*z, X + XI) ⊢
- (x+y*z, Y + XI) ⊢
- (x+y*z, x + XI) ⊢
- (+y*z, + XI) ⊢
- (y*z, XI) ⊢
- (y*z, X*YI) ⊢
- (y*z, y*YI) ⊢
- (*z, *YI) ⊢
- (z, YI) ⊢
- (z, zI) ⊢
- (ε, I)

Parsing (4) – Langkah Bottom Up

- Dorong simbol masukan saat ini ke dalam tumpukan.
- Ganti sisi kanan produksi di bagian atas tumpukan dengan sisi kiri.
- Jika bagian atas elemen tumpukan cocok dengan simbol input saat ini, lepaskan.
- Jika string input terbaca dan hanya jika simbol awal 'S' tetap ada di tumpukan, lepaskan dan lanjutkan ke status akhir 'F'.

Contoh 3

- Desain parser Bottom Up untuk ekspresi
  - x + y * z
  - P: S → S + X | X
  - X → X * Y | Y
  - Y → (S) | I

Solusi 3

- (x+y*z, I) ⊢
- (+y*z, xI) ⊢
- (+y*z, YI) ⊢
- (+y*z, XI) ⊢
- (+y*z, SI) ⊢
- (y*z, +SI) ⊢
- (*z, y+SI) ⊢
- (*z, Y+SI) ⊢
- (*z, X+SI) ⊢
- (z, *X+SI) ⊢
- (ε, z*X+SI) ⊢
- (ε, Y*X+SI) ⊢
- (ε, X+SI) ⊢
- (ε, SI)