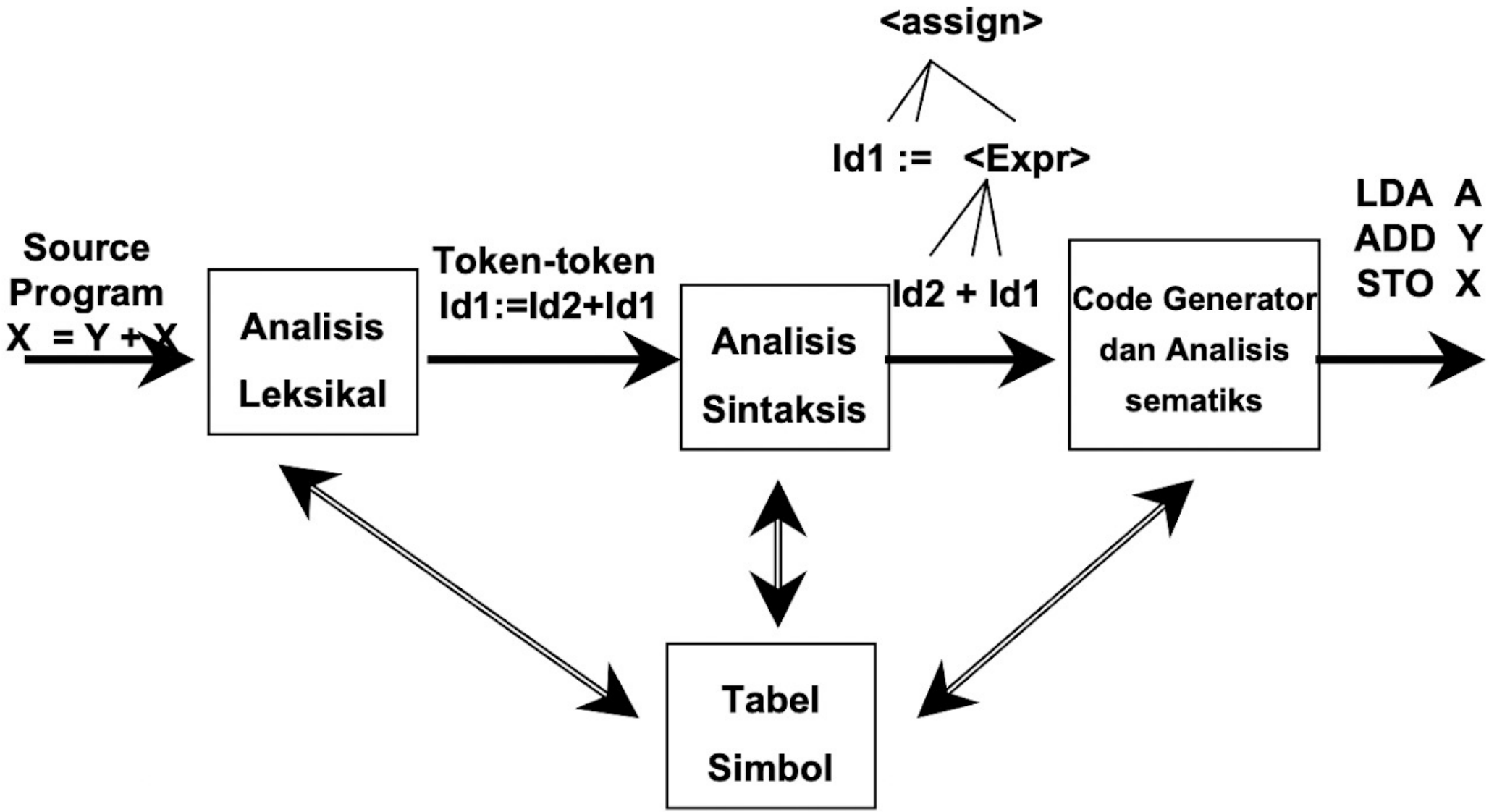
Programming Language Pragmatics

PERTEMUAN 4
REGULAR EXPRESSION
FINITE AUTOMATA

Perjalanan Sebuah Instruksi



Proses Kompilasi/ Interpretasi

1. Lexical Analysis (Regex + Automata)
   - Regex dipakai buat mendefinisikan token.
   - Automata (DFA) dipakai buat mengenali token.
2. Syntax Analysis (CFG + PDA)
   - CFG mendefinisikan aturan tata bahasa (grammar) bahasa pemrograman.
   - Parser (PDA) membangun parse tree.
3. Semantic Analysis
   - Cek makna logis (contoh: variabel sudah dideklarasi?).
   - Sudah melampaui regular/CFG → pakai logika & sistem tipe.
4. Eksekusi/ Code Generation
   - Akhirnya semua diubah ke kode mesin, dijalankan komputer (ekuivalen Turing Machine).
   - Di compiler → dibuat kode mesin.
   - Di interpreter → langsung dijalankan baris demi baris.

Chomsky Hierarchy (1/4)

Chomsky Hierarchy adalah klasifikasi bahasa formal berdasarkan kekuatannya dalam mendeskripsikan pola string. Ada 4 level utama (dari paling lemah ke paling kuat).

- ❑ Level 3: Regular Languages
  - Didefinisikan oleh: Regular Expression dan Finite State Automata (DFA/NFA)
  - Simbol: hanya terminal (misalnya huruf, digit).
  - Tidak ada non-terminal.
  - Keterbatasan: tidak bisa menghitung atau mencocokkan jumlah simbol yang "berpasangan".

Chomsky Hierarchy (2/4)

- Level 2: Context-Free Languages (CFL)
  - Didefinisikan oleh: Context-Free Grammar (CFG) dan Pushdown Automata (PDA) (automata dengan stack)
  - Simbol: Terminal: simbol konkret (a, b, +, id)
  - Non-terminal: variabel abstrak (S, E, T)
  - Lebih kuat dari regular: bisa menangani struktur rekursif atau berlapis (nested).

Chomsky Hierarchy (3/4)

- Level 1: Context-Sensitive Languages (CSL)
  - Didefinisikan oleh: Context-Free Grammar (CFG) dan Pushdown Automata (PDA) (automata dengan stack)
  - Simbol: Terminal: simbol konkret (a, b, +, id)
  - Non-terminal: variabel abstrak (S, E, T)
  - Lebih kuat dari regular: bisa menangani struktur rekursif atau berlapis (nested).

Chomsky Hierarchy (3/4)

- Level 1: Context-Sensitive Languages (CSL)
  - Didefinisikan oleh: Context-Sensitive Grammar (CSG) dan Linear Bounded Automata (LBA)
  - Aturan produksi: $\alpha A \beta \rightarrow \alpha \gamma \beta$ (panjang kanan $\geq$ panjang kiri).
  - Bisa mendeskripsikan bahasa yang membutuhkan konteks.

Chomsky Hierarchy (4/4)

- Level 0: Recursively Enumerable Languages
  - Didefinisikan oleh: Unrestricted Grammar dan Linear Bounded Automata (LBA) dan Turing Machine
  - Bisa mendeskripsikan bahasa yang membutuhkan konteks.
  - Paling umum dan kuat → bisa mendeskripsikan bahasa yang dapat dikenali oleh komputer.
  - Namun, tidak semua bahasa di level ini bisa diputuskan (halting problem).

Regular Expression (RegEx) (1/6)

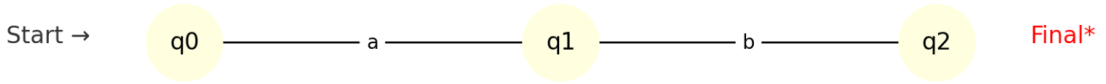
- Konkatenasi (Concatenation)

  Dua pola RegEx ditulis berdampingan → harus muncul berurutan.

  Contoh:

  - RegEx = `ab`
  - Artinya: string yang terdiri dari `"a"` diikuti `"b"`.
  - ✅ Diterima: `"ab"`
  - ❌ Ditolak: `"a"`, `"b"`, `"ba"`, `"aab"`

State Transition Automata untuk Regex: ab



- Dari **q0** (start) baca `a` → pindah ke **q1**.
- Dari **q1** baca `b` → pindah ke **q2**.
- **q2** adalah final state → string `ab` diterima.

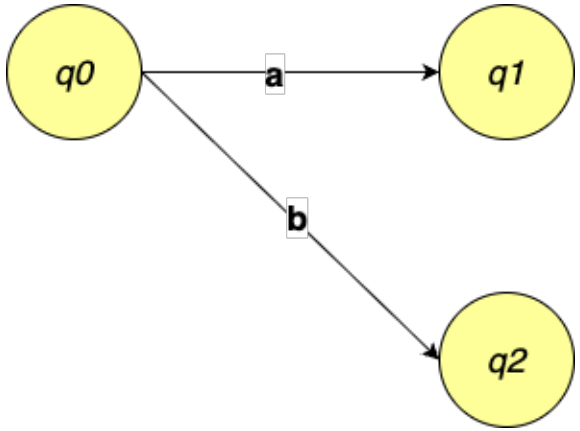
Regular Expression (RegEx) (2/6)

- Union (Alternation |)

  Simbol | berarti **atau**.

  Contoh:

  - RegEx = `a|b`
  - Artinya: string `"a"` atau `"b"`.
  - ✅ Diterima: `"a"`, `"b"`
  - ❌ Ditolak: `"ab"`, `"ba"`, `"aa"`



Lebih kompleks:

- RegEx = `(ab)|(ba)`
- ✅ Diterima: `"ab"`, `"ba"`
- ❌ Ditolak: `"a"`, `"b"`, `"aba"`

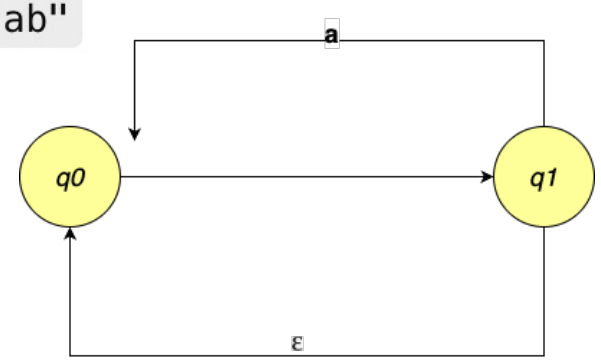
Regular Expression (RegEx) (3/6)

- Kleene Star (*)

  x* berarti **nol atau lebih** ulangan dari x.

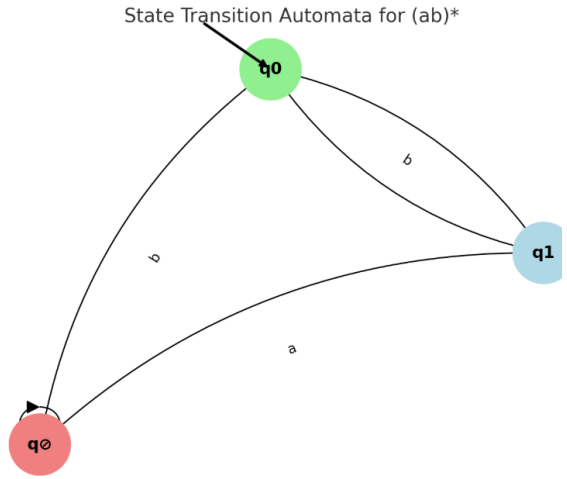
Contoh:

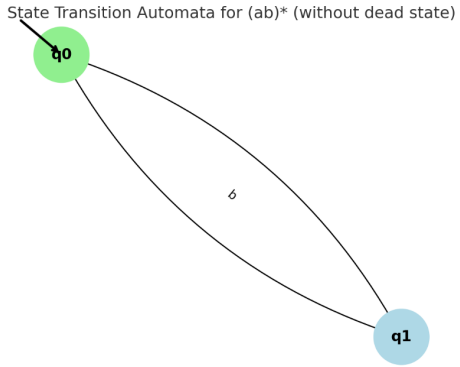
- RegEx = `a*`
- ✅ Diterima: `""` (kosong), `"a"`, `"aa"`, `"aaa"`
- ❌ Ditolak: `"b"`, `"ab"`



Contoh lain:

- RegEx = `(ab)*`
- ✅ Diterima: `""`, `"ab"`, `"abab"`, `"abababab"`
- ❌ Ditolak: `"a"`, `"b"`, `"aba"`





Regular Expression (RegEx) (4/6)

- Plus (+)

  x+ berarti **satu atau lebih** ulangan dari x.

  Contoh:

  - RegEx = `a+`
  - ✅ Diterima: `"a"`, `"aa"`, `"aaa"`, ...
  - ❌ Ditolak: `""` (kosong), `"b"`

  Contoh lain:

  - RegEx = `(ab)+`
  - ✅ Diterima: `"ab"`, `"abab"`, `"abababab"`
  - ❌ Ditolak: `""`, `"a"`, `"b"`, `"aba"`

Regular Expression (RegEx) (5/6)

- Optional (?)

  x+ berarti **satu atau lebih** ulangan dari x.

Contoh:

- RegEx = `a?`
- ✅ Diterima: `""`, `"a"`
- ❌ Ditolak: `"aa"`, `"b"`

Contoh lain:

- RegEx = `(ab)?`
- ✅ Diterima: `""`, `"ab"`
- ❌ Ditolak: `"abab"`, `"ba"`, `"a"`

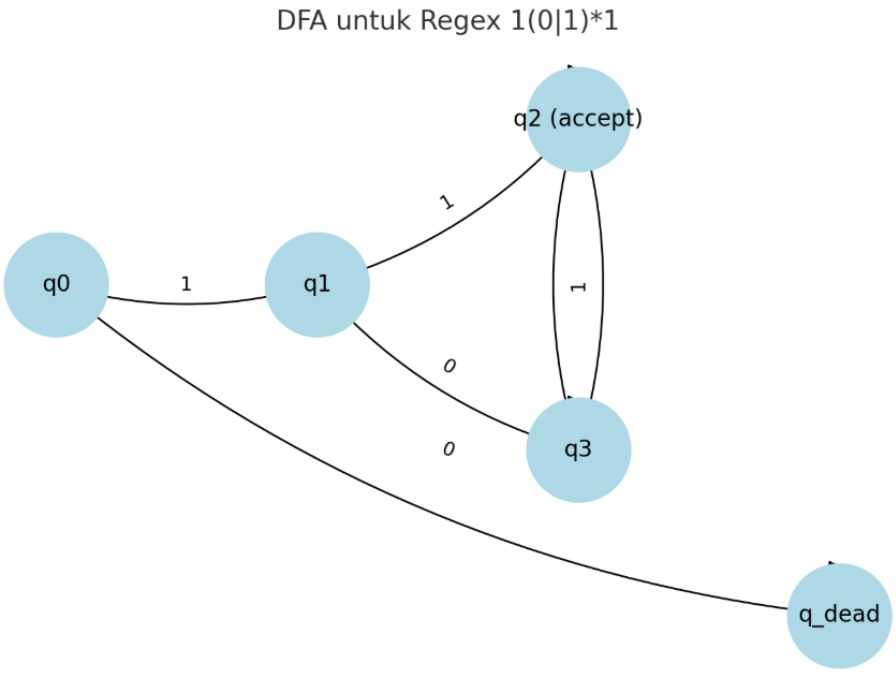
Regular Expression (RegEx) (6/6)

- Kombinasi Operasi

  Bisa digabung untuk ekspresi lebih kompleks.

Contoh:

- RegEx = `1(0|1)*1`
  - Harus **mulai dengan** `1` **dan berakhir dengan** `1`
  - ✅ Diterima: `"11"`, `"101"`, `"111"`, `"1101"`
  - ❌ Ditolak: `"0"`, `"10"`, `"100"`



Latihan Membuat State Transition Finite Automata

1. Menemukan semua angka (\d+)

   Keterangan :

   Regex: \d+
   Konstruksi (DFA sederhana):

   Alfabet Σ = {0,1,2,3,4,5,6,7,8,9, other}

   Q = {q0 (start), q1 (in-digit), q_dead}

   q0 = start; F = {q1}

   Transisi δ:

   - dari q0: digit → q1 ; other → q0 (atau tetap di q0 untuk scanning)
   - dari q1: digit → q1 ; other → q0 (kembali ke q0 menandai akhir token)
   - q_dead tidak perlu jika kita model scanning (q0 menangani non-digit)

   Penjelasan: q1 adalah state menerima yang menunjukkan sedang membaca satu atau lebih digit. Mesin bisa dijalankan pada teks untuk mengekstrak token-digit dengan memulai di q0 dan mencatat fragmen ketika q1 → q0 pada non-digit.

Latihan Membuat State Transition Finite Automata

2. **Mengenali string berakhiran "ing"**
   Regex: .*ing
   Buat automata yang menerima semua string yang diakhiri dengan huruf ing !
3. **Mengenali angka biner**
   Regex: [01]+
   Buat automata yang menerima string berisi hanya 0 dan 1, minimal satu digit !
4. **Mengenali huruf vokal tunggal**
   Regex: [aiueo]
   Buat automata yang menerima satu karakter vocal !