Compilers

INF-400

Burak Arslan
ext-inf400@burakarslan.com

Galatasaray Üniversitesi

Lecture IV
2023-10-26

Course website

burakarslan.com/inf400

First assignment is due next week!

Deadline: **2023-11-01 23:59**

Submission format

A file named `<student-id>.tar.xz` that contains:

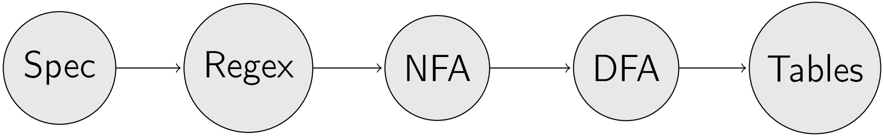
- A file named `sol1.py` that contains the expected python script.
- A directory named `sol2` that contains the lexical analyzer with an optional `hw1.{md,pdf}` with anything noteworthy.

Tips on Building Large Systems

Compilers are big projects with many moving parts.

- KISS (Keep It Simple, Stupid!)
- Don't optimize prematurely
- Design systems that can be tested (and test them!)
- It is easier to modify a working system than to get a system working

Implementing Lexers



Implementing Lexers

Regular Expressions

Remember: Regular expressions specify sets of strings.

$\forall A, B \in \textcircled{S}$, sets of strings over alphabet Σ;

- Neutral: $\{\texttt{""}\} \Longrightarrow \varepsilon$[1] $\neq \varnothing$
- Union: $A \cup B \Longrightarrow (A|B)$
- Concatenation: $\{s_1 s_2 \mid s_1 \in A \ \wedge \ s_2 \in B\} \Longrightarrow AB$
- Range: $\{\texttt{"a"}, \texttt{"b"}, \ldots, \texttt{"z"}\} \Longrightarrow [\mathrm{a} - \mathrm{z}]$
- Range Excl.: $\textcircled{S} \setminus \{\texttt{"a"}, \texttt{"b"}, \ldots, \texttt{"z"}\} \Longrightarrow$ `[^a-z]`

[1] singleton with an empty string

Implementing Lexers

Regular Expressions

Repetitions:

Let $A \in Ⓢ$, sets of strings over alphabet Σ, $A^n = \underbrace{AA \dots A}_{n}$

- Optional: $A + \varepsilon \implies A?$
- Zero or more: $\bigcup_{i \geq 0} A^i \implies A*$
- One or more: $\bigcup_{i > 0} A^i \implies A+$
- Explicit:
  - $\bigcup_{i \geq n} A^i \implies A\{n\}$
  - $\bigcup_{i \geq n, i \leq m} A^i \implies A\{n, m\}$ where $n \leq m$

Implementing Lexers

Regular expressions

are implemented using

Finite State Automata

Implementing Lexers

Finite State Automata

Two types:

- **DFA**: Deterministic Finite Automata
- **NFA**: Nondeterministic Finite Automata

Implementing Lexers

Finite State Automata

Two types:

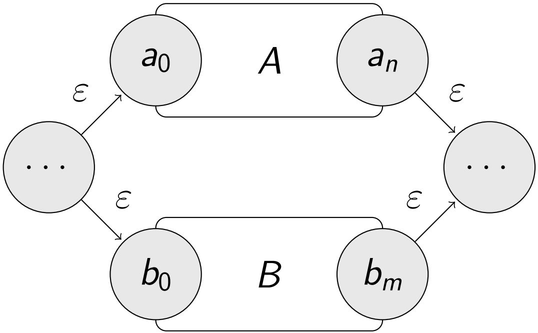
- **DFA**:
  - No more than one move per input
  - ε moves are forbidden
- **NFA**:
  - Zero or more moves per input
  - ε moves are allowed

Implementing Lexers

Finite State Automata

Regular expressions have direct NFA representations.

Eg. `(A|B)`:

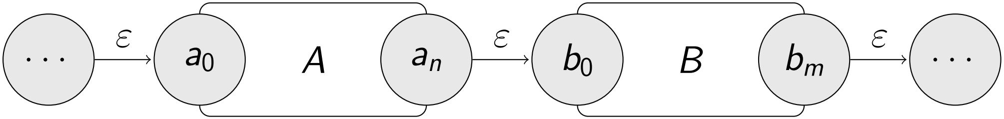


Implementing Lexers

Finite State Automata

Regular expressions have direct NFA representations.

Eg. `AB`:



Implementing Lexers

Finite State Automata

NFA and DFA are equivalent and recognize both **regular languages**.

DFAs are faster to execute

Implementing Lexers

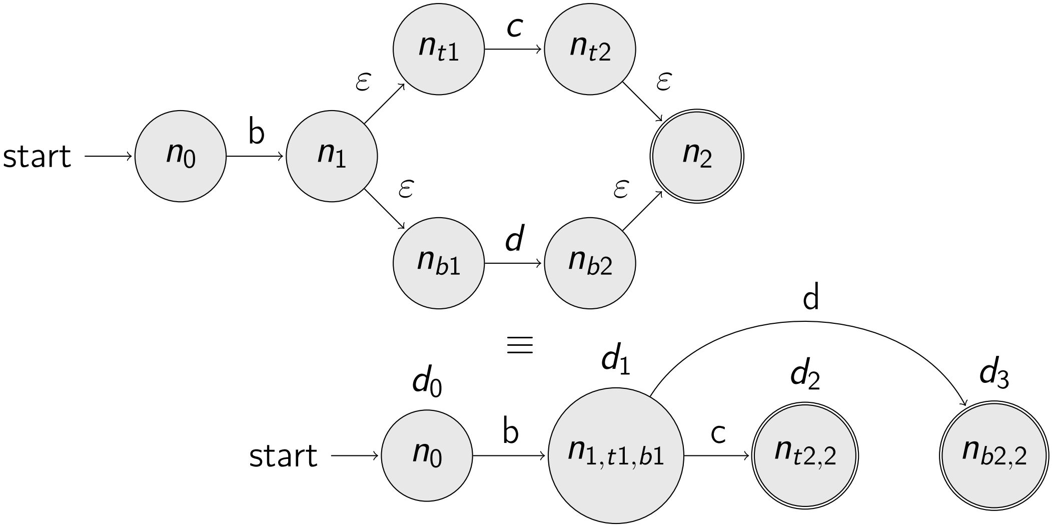
Finite State Automata

Conversion algorithm: **Simulation** / **Tracing** (recording execution)

- Start state of the DFA = States reachable from the start state of the NFA through ε input
- Let $n, n', n'', \ldots$ states from NFA,
- Let $d, d', d'', \ldots$ states from DFA,
- Add a new state $d \underset{a}{\rightarrow} d'$ if and only if n' is reachable from n, including ε input

Implementing Lexers

Finite State Automata



Implementing Lexers

Implementation

We use tables / grids / 2D arrays:

	input		
	b	c	d
d_0	d_1		
d_1		d_2	d_3
d_2			
d_3			

Implementing Lexers

Implementation

All in all, lexer generators' job boil down to:

- Unify all regular expressions into a single NFA
- Perform NFA $\Rightarrow$ DFA conversion
- Create DFA grid
- Execute DFA