Finite Automata

COP-3402 Systems Software

Paul Gazzillo

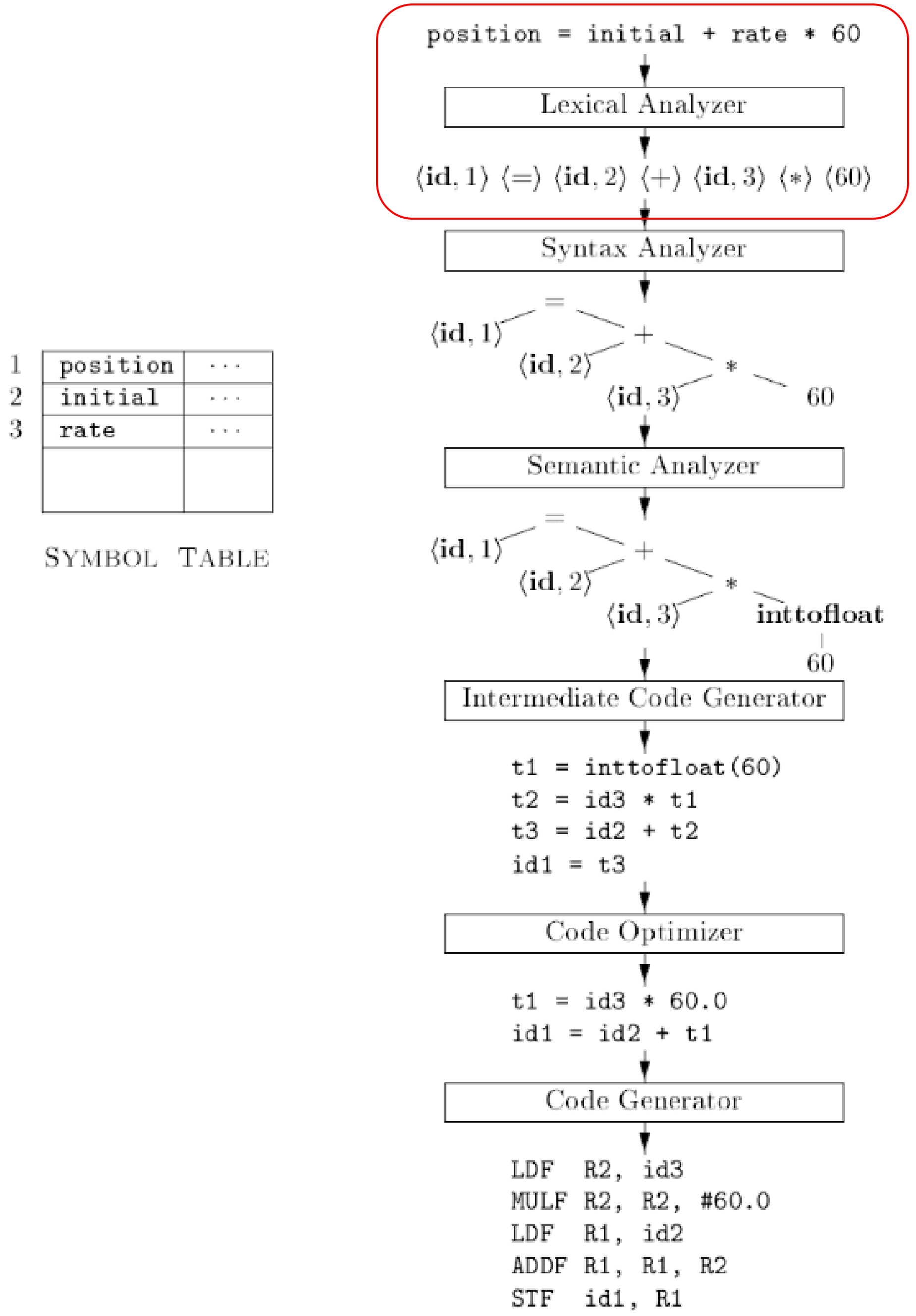


Figure 1.7: Translation of an assignment statement

Token	Informal Description	Sample Lexemes
if	characters `i`, `f`	`if`
else	characters `e`, `l`, `s`, `e`	`else`
comparison	`<` or `>` or `<=` or `>=` or `==` or `!=`	`<=`, `!=`
id	letter followed by letters and digits	`pi`, `score`, `D2`
number	any numeric constant	`3.14159`, `0`, `6.02e23`
literal	anything but `"`, surrounded by `"`'s	`"core dumped"`

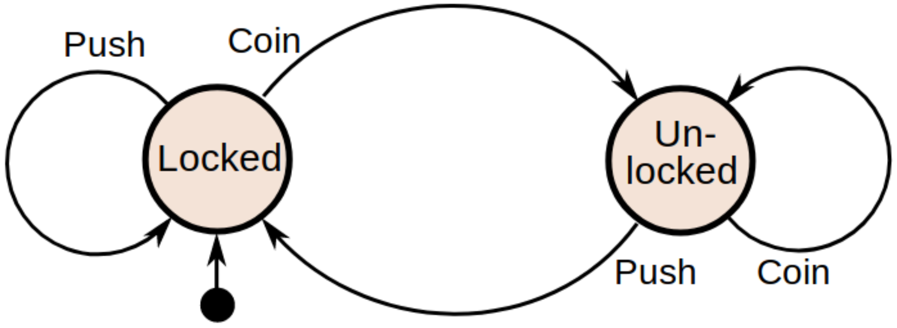
Figure 3.2: Examples of tokens

Core Problem: Recognizing Tokens

- Regular expressions *specify* tokens
- How do we recognize them?
- Use *finite automata*, a.k.a., finite-state machines

Finite Automata are a Model of Computation

- A model of computation: transitions between states
  - Finite alphabet of symbols (just like regular languages)
  - Finite set of states
  - Set of accepting states
  - An initial state
  - A transition function: f(current_state, symbol) -> next_state



https://en.wikipedia.org/wiki/Finite-state_machine#Example:_coin-operated_turnstile

Finite Automata Have Wide Application

- Vending machines: count coins
- Elevators: sequence of stops
- Traffic lights: order of changes
- Combination lock: numbers in correct order

Representation with Transition Diagrams

- States: circles
- Transitions: labeled arrows
- Starting state: arrow
- Accepting states: double circles

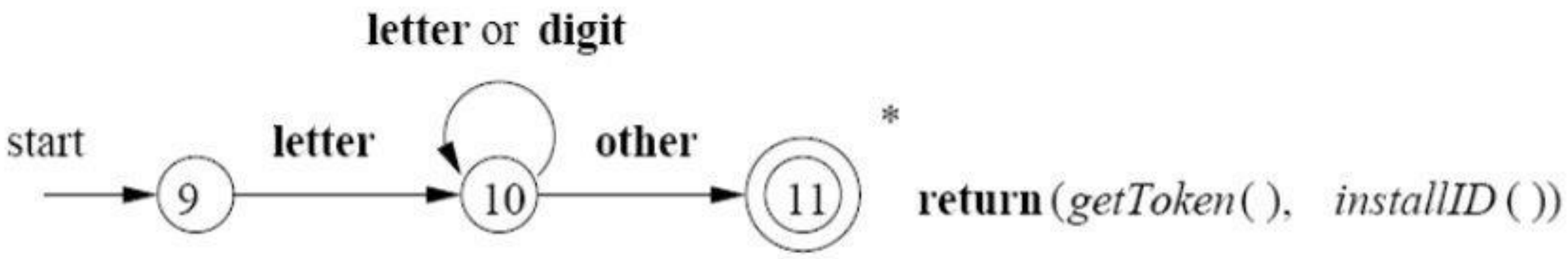
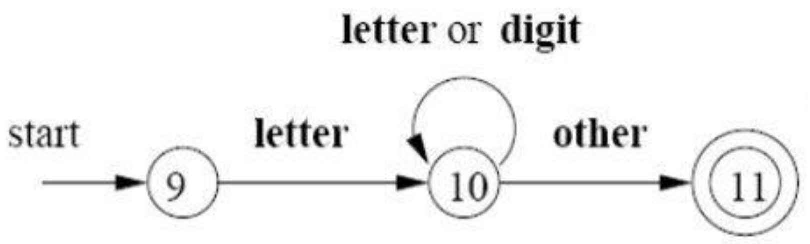


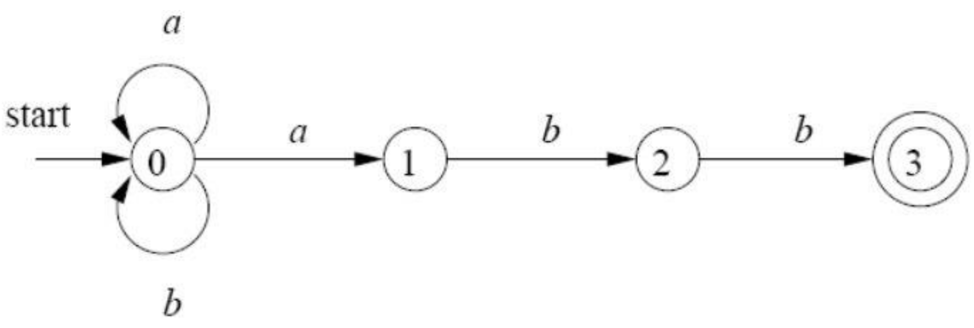
Figure 3.14: A transition diagram for **id**'s and keywords

Deterministic vs Nondeterministic Finite Automata (DFA vs NFA)

- *Deterministic:* one state at-a-time
  - We can uniquely "determine" which state we are in



- *Nondeterministic:* in multiple states at once
  - Must track all states *simultaneously*



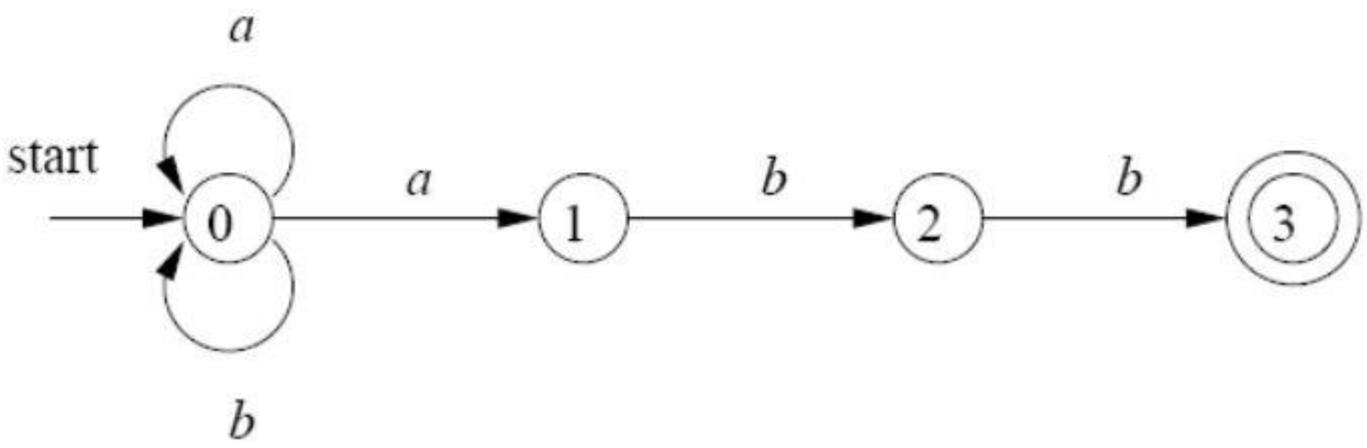


Figure 3.24: A nondeterministic finite automaton

STATE	a	b	ϵ
0	$\{0,1\}$	$\{0\}$	$\emptyset$
1	$\emptyset$	$\{2\}$	$\emptyset$
2	$\emptyset$	$\{3\}$	$\emptyset$
3	$\emptyset$	$\emptyset$	$\emptyset$

Figure 3.25: Transition table for the NFA of Fig. 3.24

Remember: Epsilon (ε) is Empty String

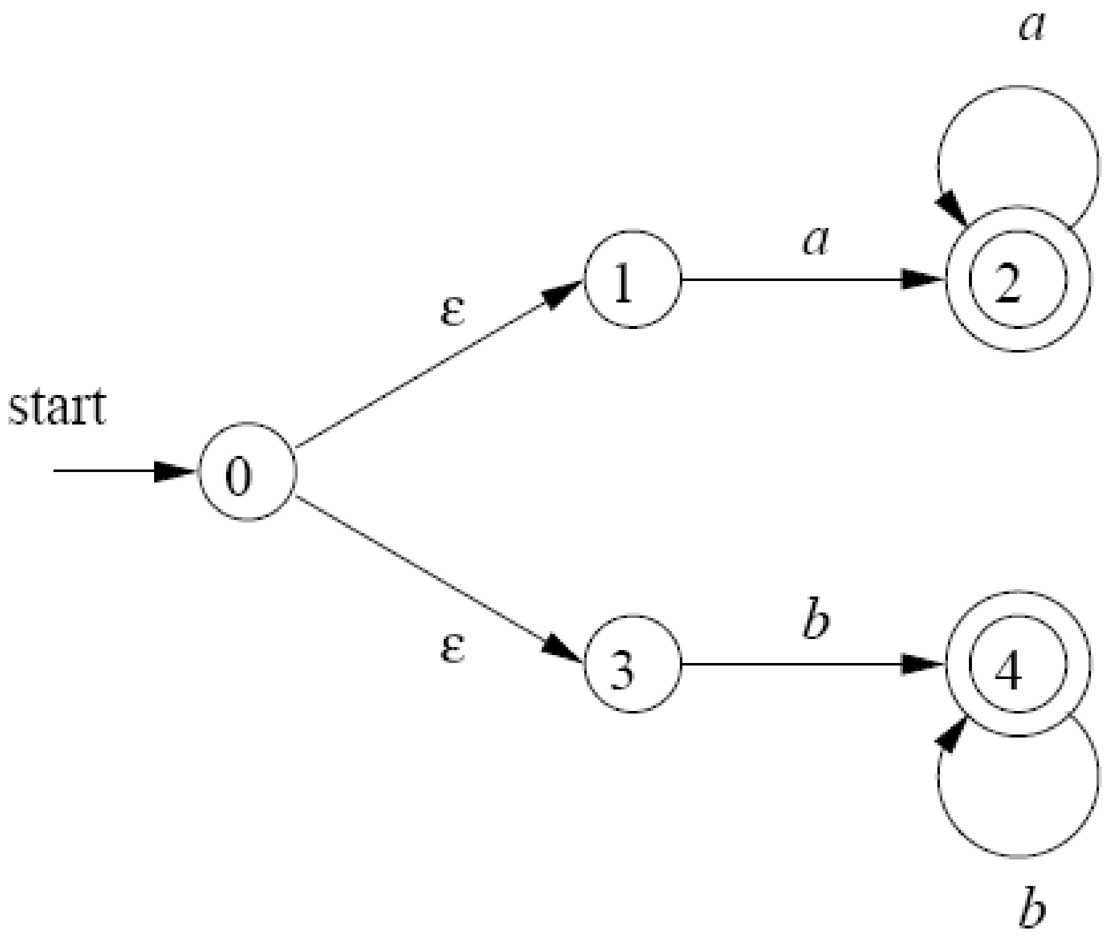
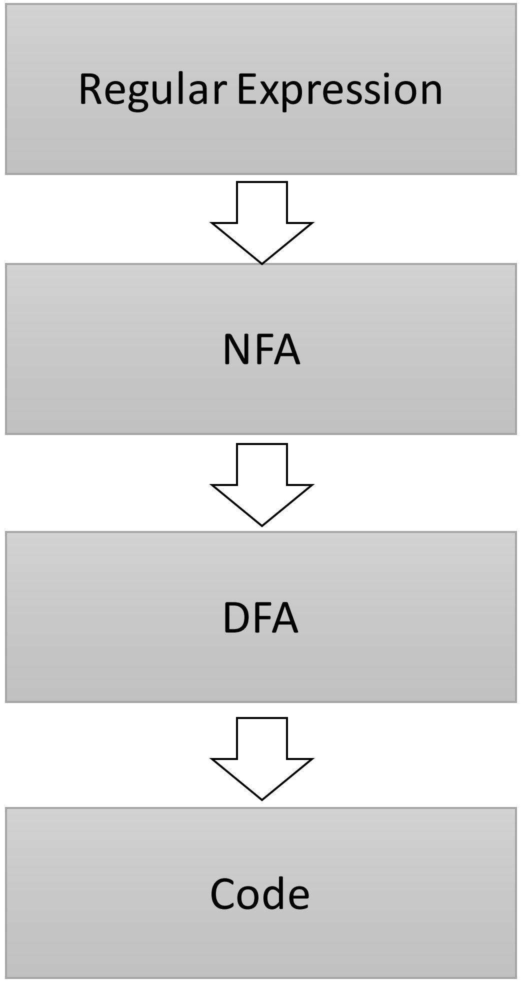


Figure 3.26: NFA accepting $\mathbf{aa}^*|\mathbf{bb}^*$

Properties of Finite Automata

- Many possible NFAs/DFAs for same language
- There is a provably minimal DFA for a language
- All DFAs are also (degenerate) NFAs

Using Finite Automata to Implement a Lexer



Regular Expression to NFA

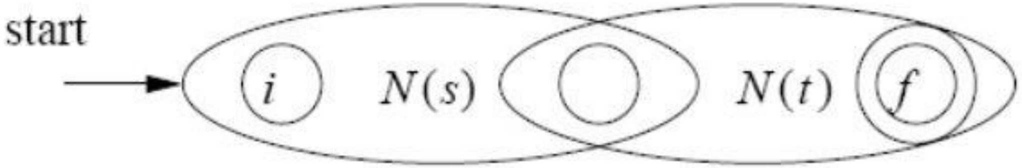


Figure 3.41: NFA for the concatenation of two regular expressions

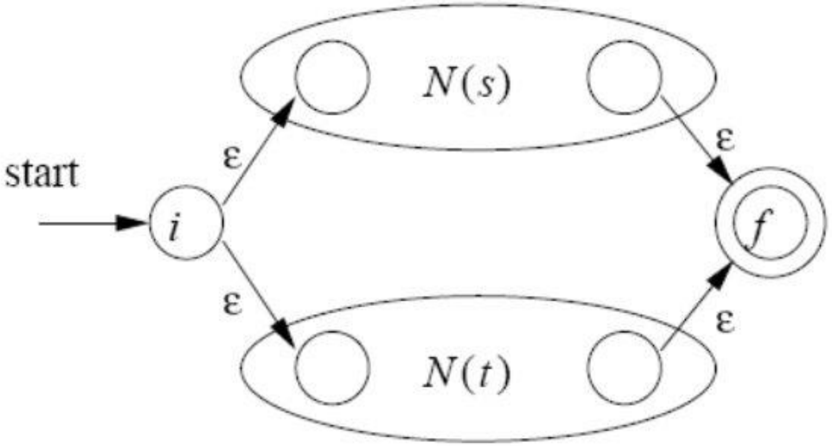


Figure 3.40: NFA for the union of two regular expressions

Regular Expression to NFA (continued)

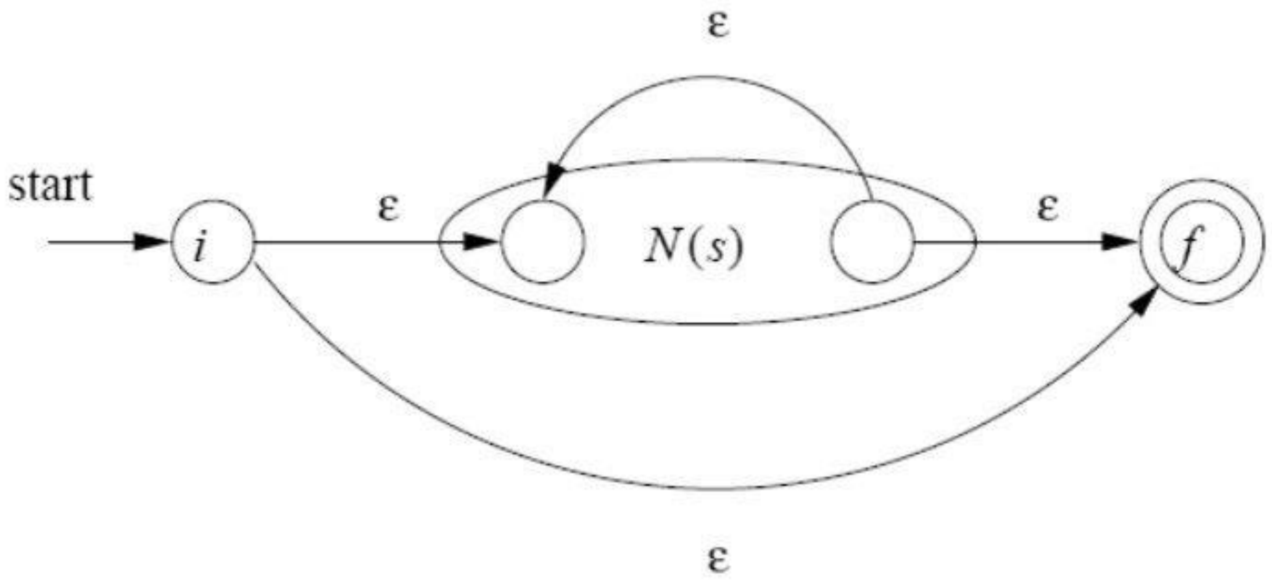


Figure 3.42: NFA for the closure of a regular expression

NFA to DFA with Subset Construction

- Construct DFA from NFA systematically
- Each DFA state created from subset of NFA states
  - Remember: can be in multiple states
- "Simulate" being in multiple states using a single state
- Dragon book 3.7

Example

Converting regular expressions to nondeterministic and deterministic automata