Lexical Analysis

Regular Expressions

Nondeterministic Finite Automata (NFA)

Deterministic Finite Automata (DFA)

Implementation Of DFA

Regular Expressions (REs)

- Compact mechanism for defining a language
  - Generally easier to understand than FSMs
- Example: identifier – letter followed by zero or more letters or digits

  letter (letter|digit)*
- Used as input to scanner generator
  - Define each token, also
  - Define white-space, comments, etc.
    - Things that do not correspond to tokens but must also be recognized and ignored

Regular Expression Operators

X Y	concatentation	X followed by Y
X | Y	alternation	X or Y (alternatives)
X *	Kleene closure	Zero or more occurrences of X
X +		One or more occurrence of X
(X)	grouping	Used for grouping (as in programming languages)

Operands of RE Operators

- The empty string ε
- Single characters of the underlying alphabet
- Shorthands for groups of characters (letter for A-Z or a-z, digit for 0-9, etc.)
- Legal regular expressions (an operator may be applied to the result of an operator)

Precedence for RE Operators

Regular Expression Operator	Analagous Arithmetic Operator	Precedence
X | Y	X + Y	lowest
X Y	X * Y	middle
X *, X +	X ^ Y	highest

- For example:
  letter letter | digit *
  letter (letter | digit) *

Language Defined By a RE

- Recall, for an automaton the language is the set of strings accepted by the automaton
- For a RE, the language is the set of strings *matched* by the RE

Regular Expression	Set of Strings
ε	{ "" }
ab	{ "ab" }
a | b | c	{ "a", "b", "c" }
(a | b | c) *	{ "", "a", "b", "c", "aa", "ab", "ac", "ba", "bb", "bc", … }

Understanding REs

- Describe these languages:

  0 (0 | 1)* 0

  (0 | 1) * 0 (0 | 1) (0 | 1)

  1* (0 (1 *) 0 (1 *)) *

Writing Regular Expressions

- Translate these into regular expressions
  - Words ending in “ing” (a word consists of lower or upper-case letters)
  - Binary strings with an odd number of 1s

Writing Regular Expressions

- Floating point numbers with an optional leading sign (+ or -) consisting of at least one digit and an optional decimal point (if there is a decimal point, there must be at least one digit before and one after the decimal point)

From RE to a Scanner

- Theorem: for every regular expression, there is a deterministic finite-state machine that defines the same language (and vice versa)
- Q: How do we create this machine (automatically)?
- Idea: start by translating a RE to an NFA

Lexical Analysis

Regular Expressions

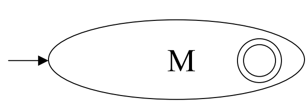
Nondeterministic Finite Automata (NFA)

Deterministic Finite Automata (DFA)

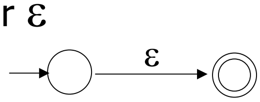
Implementation Of DFA

RE to NFA(1)

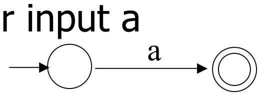
- For each kind of RE, define an NFA
  - Notation: NFA for RE M



  - For ε

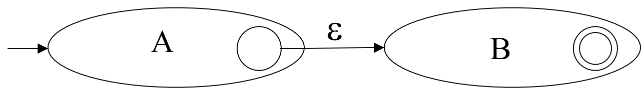


  - For input a

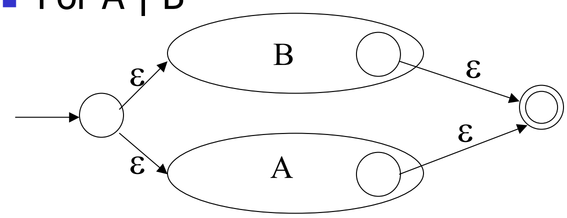


RE to NFA (2)

- For A B

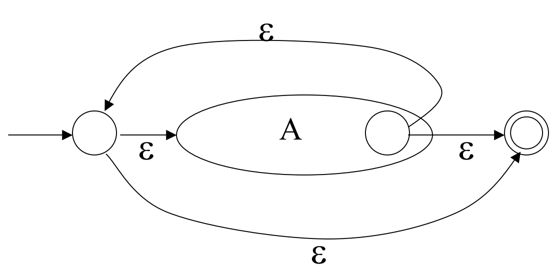


- For A | B



RE to NFA (3)

- For A*



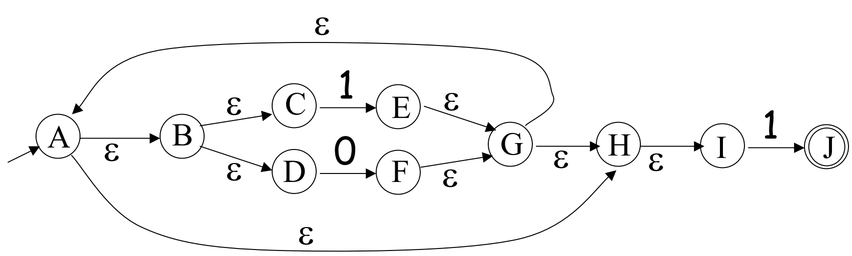
- A+ ?

Example: RE to NFA

- Consider the regular expression

  (1|0)*1

- The NFA is



Another Example

(ε | + | -) digit+ (ε | . digit+)

Lexical Analysis

Regular Expressions

Nondeterministic Finite Automata (NFA)

Deterministic Finite Automata (DFA)

Implementation Of DFA

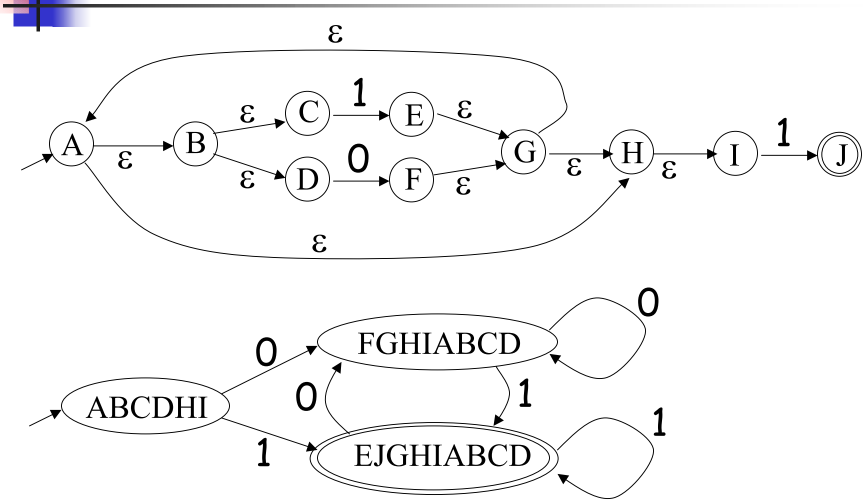
NFA to DFA: The Trick

- Simulate the NFA
- Each state of the DFA
  = a non-empty subset of states of the NFA
- Start state
  = the set of NFA states reachable through ε-moves from NFA start state
- Add a transition $S \rightarrow^{a} S'$ to DFA iff
  - S' is the set of NFA states reachable from any state in S after seeing the input a, considering ε-moves as well

NFA to DFA: Remark

- An NFA may be in many states at any time
- How many different states ?
- If there are N states, the NFA must be in some subset of those N states
- How many subsets are there?
- $2^N - 1$ = finitely many

NFA -> DFA Example



NFA to DFA: the practice

- NFA to DFA conversion is at the heart of tools such as flex
- But, DFAs can be huge
- In practice, flex-like tools trade off speed for space in the choice of NFA and DFA representations