# Lexical Analysis

Prof. James L. Frankel
Harvard University

Version of 5:37 PM 30-Jan-2018

# *Regular Expression* Notation

- We will develop a notation – called *regular expressions* – to describe languages that can be built from an alphabet (letters, numbers, and other symbols) with certain operations applied – potentially recursively applied

- Operators are
  - Union – designated by infix | (vertical bar)
  - Concatenation – designated by sequential subexpressions
  - Closure – designated by a postfix superscript of either * or +

# Alphabet & Strings

- An *alphabet* is any finite set of symbols
- A *string* over an alphabet is a finite sequence of symbols drawn from that alphabet
- $|s|$ is the length of string *s*, *i.e.,* the number of symbols in string *s*
- ε is the string of length zero, *i.e.,* the *empty string*

# Language, Union, and Concatenation

- Assume L is a *language*
  - L is any countable set of strings over some fixed alphabet
  - Includes the empty set, syntactically well-formed C programs, all grammatically correct English sentences
  - This definition of *language* does not ascribe any meaning to strings in the language
- Union of languages
  - Identical to the set operation of union
  - A language formed from the **union** of two languages is composed of all strings in either the first language or the second language
- Concatenation of languages
  - A language formed from the **concatenation** of two languages is composed of all strings formed by taking a string from the first language and a string from the second language, in all possible ways, and concatenating them

# Closure

- Kleene Closure or, simply, Closure
  - The set of strings that can be created by concatenating L **zero** or more times
    - Denoted by $\mathbf{L^*}$
  - $L^0$ (*i.e.,* concatenation of L zero times) is defined to be { ε }
  - $L^i$ , for i > 0, is defined to be $L^{i-1}$ L
- Positive Closure
  - Same as the Kleene Closure, but without $L^0$
    - The set of strings that can be created by concatenating L **one** or more times
    - Denoted by $\mathbf{L^+}$
  - ε is not in $L^+$ unless ε is in L itself

# Regular Expression

- Σ is the alphabet
- L(*r*), where *r* is a regular expression, is the language denoted by *r*
- ε is a regular expression
- L(ε) is {ε}, that is, the language whose sole member is the empty string
- If *a* is a symbol in Σ, then
  *a* is a regular expression and
  L(*a*) = {*a*}
- For regular expressions *r* and *s*,
  - (*r*) is a regular expression denoting L(*r*)
  - (*r*)|(*s*) is a regular expression denoting L(*r*) U L(*s*)
  - (*r*)(*s*) is a regular expression denoting L(*r*)L(*s*)
  - (*r*)* is a regular expression denoting (L(*r*))*

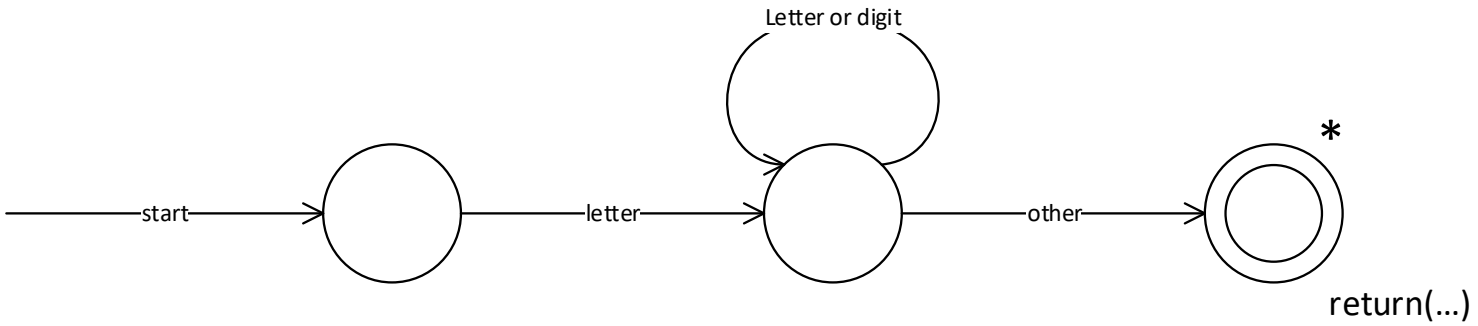
# Precedence of Regular Expression Operators

- Highest precedence: Closure (or unary *)
- Next highest precedence: Concatenation
- Lowest precedence: Union or Alternation (or |)
- All operators are left associative

# Algebraic Laws for Regular Expressions

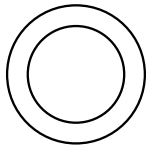
| | |
|---|---|
| • r \| s = s \| r | \| is commutative |
| • r \| (s \| t) = (r \| s) \| t | \| is associative |
| • r(st) = (rs)t | concatenation is associative |
| • r(s \| t) = rs \| rt | concatenation distributes over \| |
| • (s \| t)r = sr \|tr | concatenation distributes over \| |
| • εr = rε = r | ε is the identity for concatenation |
| • r* = (r \| ε)* | ε is guaranteed in a closure |
| • r** = r* | * is idempotent |

# Transition Diagrams





Designated start state



Accepting (final) state

Edges labeled with symbol or set of symbols

* Retract one position (symbol)

# Nondeterministic Finite Automata (NFA)

- A finite set of states *S*.
- A set of input symbols Σ, the *input alphabet*. We assume that ε, which stands for the empty string, is never a member of Σ.
- A *transition function* that gives, for each state, and for each symbol in Σ U {ε} a set of *next states*.
- A state $s_0$ from *S* that is distinguished as the *start state* (or *initial state*).
- A set of states *F*, a subset of *S*, that is distinguished as the *accepting states* (or *final states*).

# Nondeterministic Finite Automata (NFA) compared to Deterministic Finite Automata (DFA)

- NFA
  - No restrictions on the labels on edges
    - The same symbol can label several edges out of the same state
    - ε, the empty string, is also a possible label
- DFA
  - For each state, and for each symbol of its input alphabet, there can be exactly one edge with that symbol leaving that state
  - No edge can be labeled with ε, the empty string
- Either an NFA or a DFA can be represented by a *transition graph*

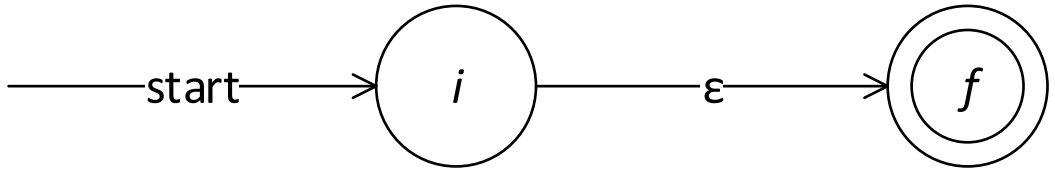
# Construction of an NFA from a Regular Expression

- Apply the McNaughton-Yamada-Thompson algorithm

- Apply the algorithm on constituent subexpressions

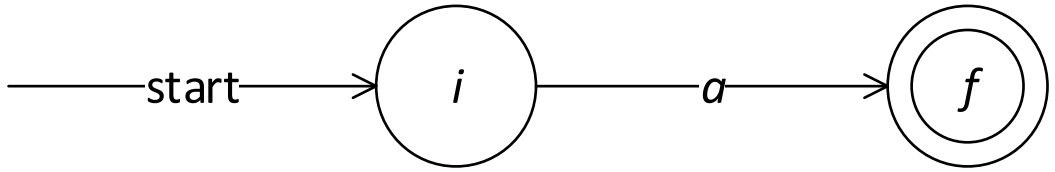
# Construction of an NFA: expression ε

- For regular expression *r* = ε



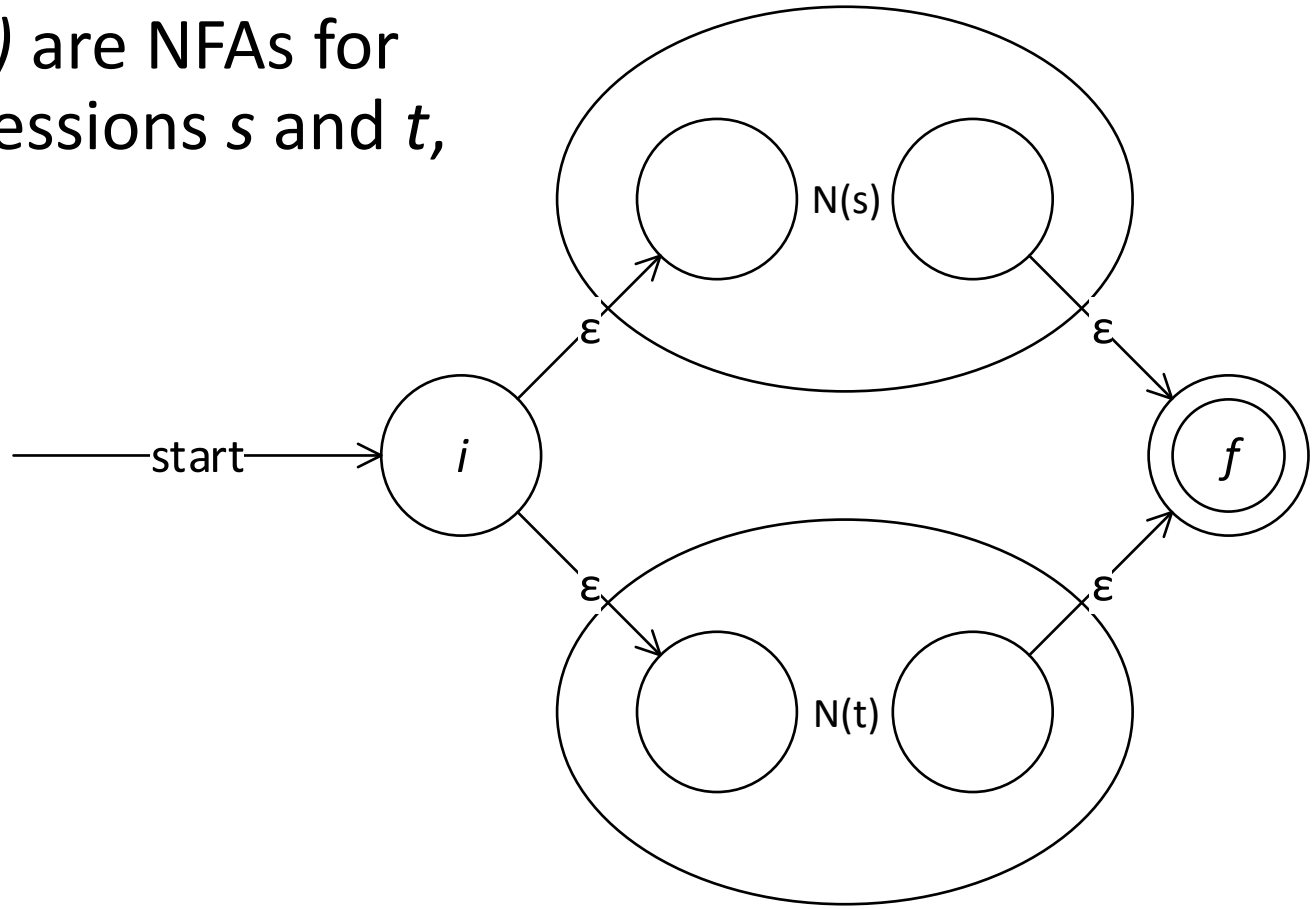
# Construction of an NFA: expression a in Σ

- For regular expression *r* = *a* (in Σ)



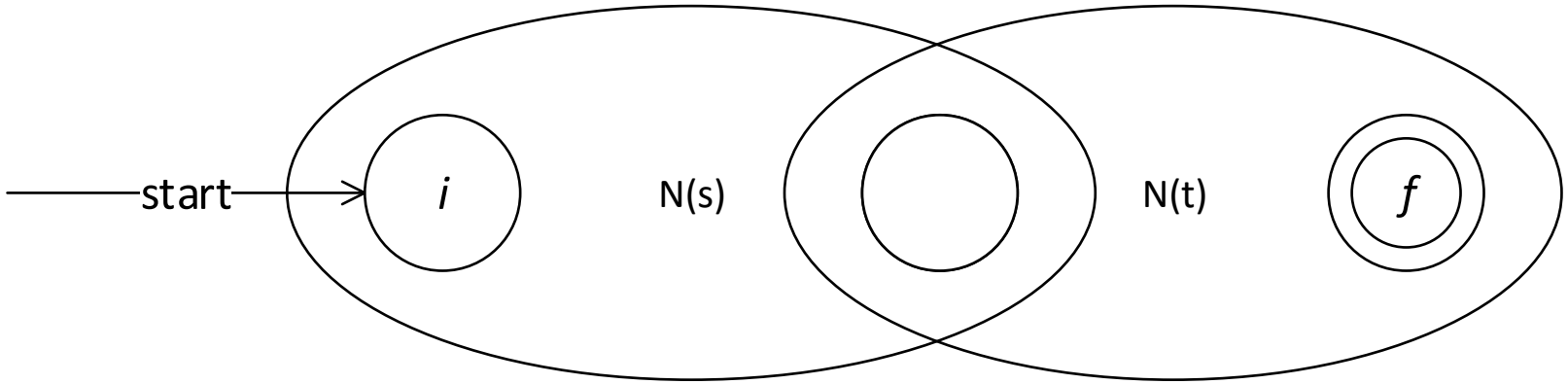
# Construction of an NFA: expression with union (that is, |)

- For regular expression $r = s \mid t$
- *N(s)* and *N(t)* are NFAs for regular expressions *s* and *t*, respectively



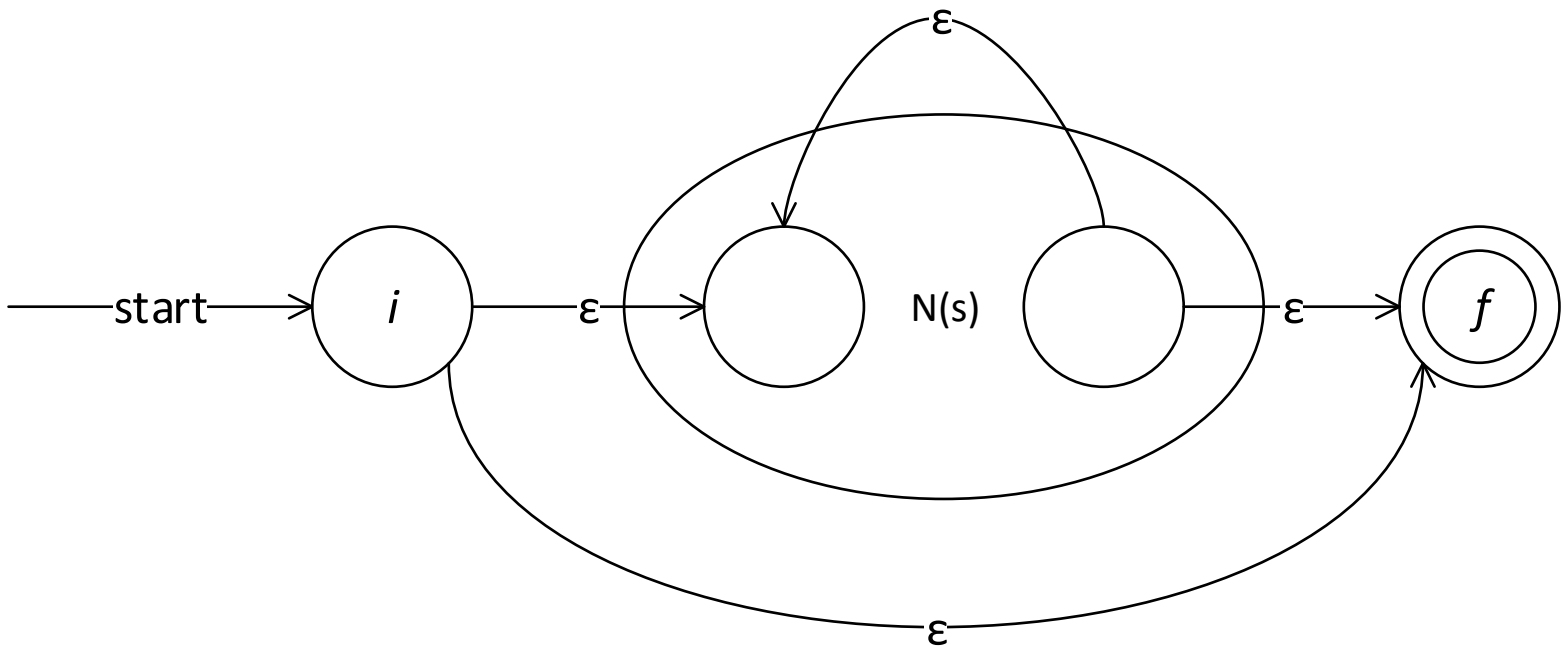
# Construction of an NFA: expression with concatenation

- For regular expression *r* = *st*
- *N(s)* and *N(t)* are NFAs for regular expressions *s* and *t*, respectively



# Construction of an NFA: expression with closure (that is, *)

- For regular expression *r* = *s**
- *N(s)* is an NFA for regular expression *s*



# Subset Construction of a DFA from an NFA

- We refer to the NFA as N and to the DFA as D
- D's states will be called *Dstates*
- D's transitions will be encoded in a transition table *Dtran*
- Each state of D is a *set* of NFA states

# Subset Construction of a DFA from an NFA

- s refers to a single state of N
- T refers to a set of states of N

- ε-closure(s) is the set of NFA states reachable from NFA state s on ε-transitions alone
- ε-closure(T) is the set of NFA states reachable from some NFA state s in set T on ε-transitions alone
- move(T, a) is the set of NFA states to which there is a transition on input symbol a from some state in T

# Computing ε-closure(T)

```
push all states of T onto stack;
initialize ε-closure(T) to T;
while ( stack is not empty ) {
  pop t, the top element, off stack;
  for ( each state u with an edge from t to u labeled ε )
    if ( u is not in ε-closure(T) ) {
      add u to ε-closure(T);
      push u onto stack;
    }
}
```

# Subset construction

initially, ε-closure($s_0$) is the only state in *Dstates*, and it is unmarked;
while ( there is an unmarked state T in *Dstates* ) {
  mark T;
  for ( each input symbol a ) {
    U = ε-closure(move(T, a));
    if ( U is not in *Dstates*)
      add U as an unmarked state in *Dstates*;
    *Dtran*[T, a] = U;
  }
}

# Example

- Construct an NFA from a regular expression
- Construct a DFA from an NFA