Lexical Analysis
Programming Languages

Sujit Kumar Chakrabarti

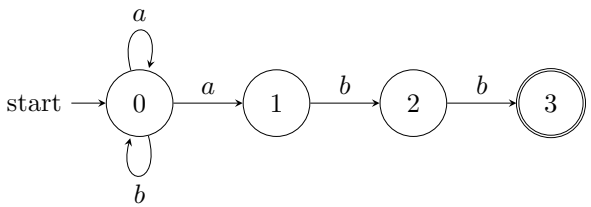
IIITB

Finite State Automata (FSA)

1. Non-deterministic FSA
2. Deterministic FSA

Non-Deterministic FSA (NFA)

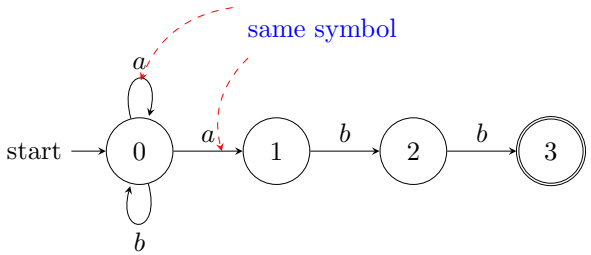
Example 1



Language: $(a|b) \star abb$

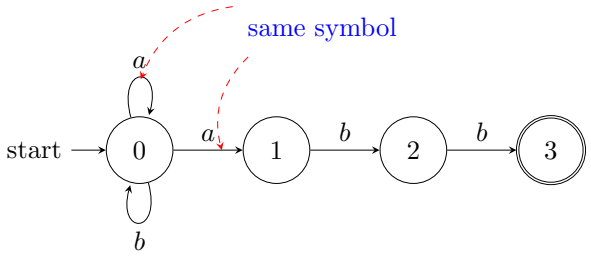
Non-Deterministic FSA (NFA)

Example 1



Non-Deterministic FSA (NFA)

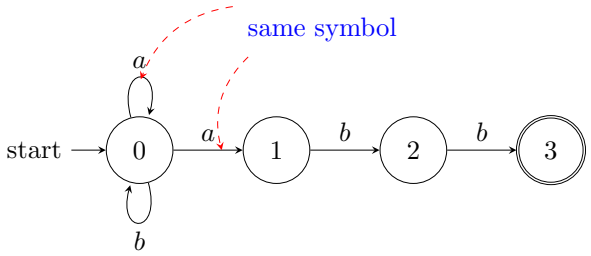
Example 1



Language:

Non-Deterministic FSA (NFA)

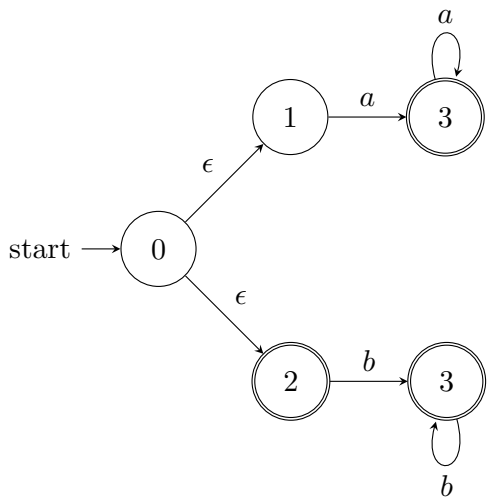
Example 1



Language: $(a|b) \star abb$

Non-Deterministic FSA (NFA)

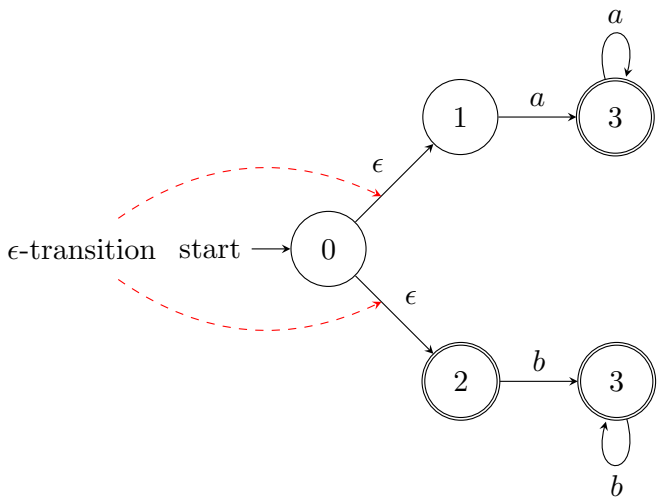
Example 2



Language: $aa\star|bb\star$

Non-Deterministic FSA (NFA)

Example 2



Language: $aa\star|bb\star$

Non-Deterministic FSA (NFA)

- Finite set of states – (S)
- Alphabet - $(\sum)$
- Transition function $(T : S \times \sum \rightarrow 2^S)$
- Initial state (S_0)
- Final/accepting states $(F \subseteq S)$

Non-Deterministic FSA (NFA)

- Finite set of states – (S)
- Alphabet - $(\sum)$
- Transition function $(T : S \times \sum \rightarrow 2^S)$
- Initial state (S_0)
- Final/accepting states $(F \subseteq S)$
- **Acceptance of a string:** When there exists a path corresponding to the input leading to an accepting state.

Non-Deterministic FSA (NFA)

- Finite set of states – (S)
- Alphabet - $(\sum)$
- Transition function $(T : S \times \sum \rightarrow 2^S)$
- Initial state (S_0)
- Final/accepting states $(F \subseteq S)$
- **Acceptance of a string:** When there exists a path corresponding to the input leading to an accepting state.

Specific Properties

- The same state can transition to more than one states on the same symbol
- ϵ-transitions

Deterministic FSA (DFA)

- Finite set of states – (S)
- Alphabet - $(\sum)$
- Transition function $(T : S \times \sum \rightarrow S)$
- Initial state (S_0)
- Final/accepting states $(F \subseteq S)$
- **Acceptance of a string:** When there exists a path corresponding to the input leading to an accepting state.

Deterministic FSA (DFA)

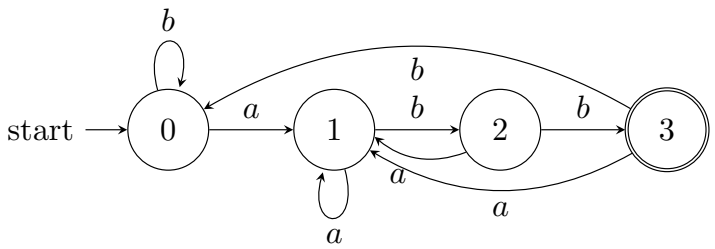
- Finite set of states – (S)
- Alphabet - $(\sum)$
- Transition function $(T : S \times \sum \rightarrow S)$
- Initial state (S_0)
- Final/accepting states $(F \subseteq S)$
- **Acceptance of a string:** When there exists a path corresponding to the input leading to an accepting state.

Specific Properties

- Only one next-state on the same symbol
- No ϵ-transitions

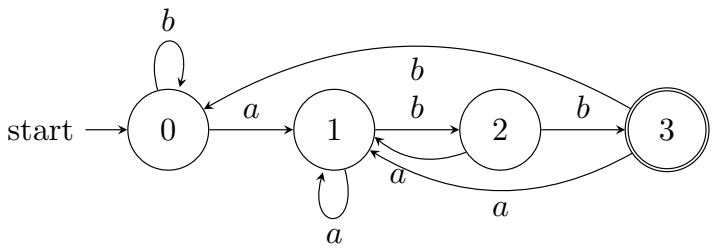
Deterministic FSA (DFA)

Example 1



Deterministic FSA (DFA)

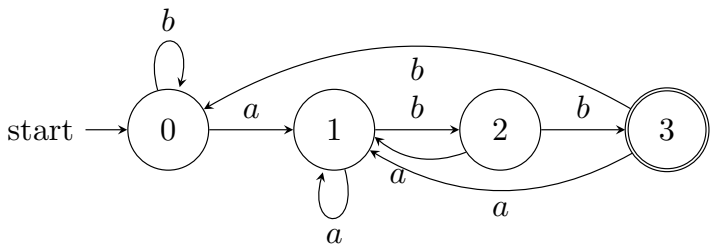
Example 1



Language:

Deterministic FSA (DFA)

Example 1



Language: $(a|b) \star abb$

NFA and DFA

- NFAs: Often more readable
- NFAs: Usually have fewer states

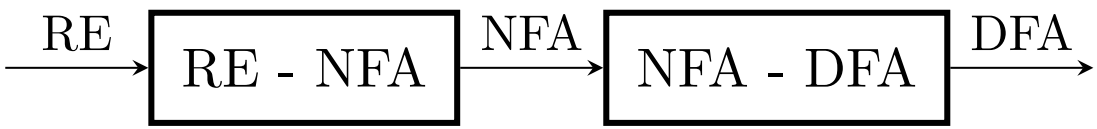
NFA and DFA

- NFAs: Often more readable
- NFAs: Usually have fewer states
- DFAs: Less readable
- DFAs: Larger number of states
- DFAs: Faster to simulate

NFA and DFA

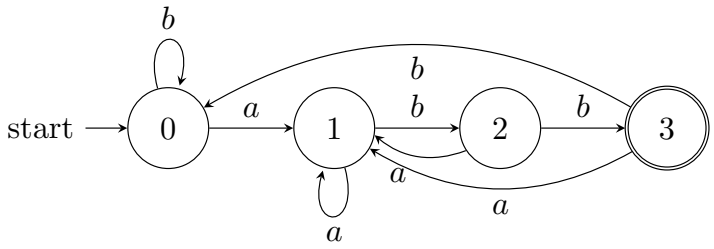
- NFAs: Often more readable
- NFAs: Usually have fewer states
- DFAs: Less readable
- DFAs: Larger number of states
- DFAs: Faster to simulate
- Equally expressive $\equiv$ Regular expressions (Regular languages)

Lexical Analysis Process



Simulating FSAs

Representating transition function using transition tables

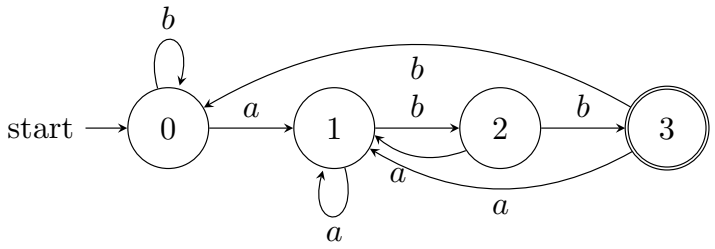


Transition Table:

State	a	b
0		
1		
2		
3		

Simulating FSAs

Representating transition function using transition tables

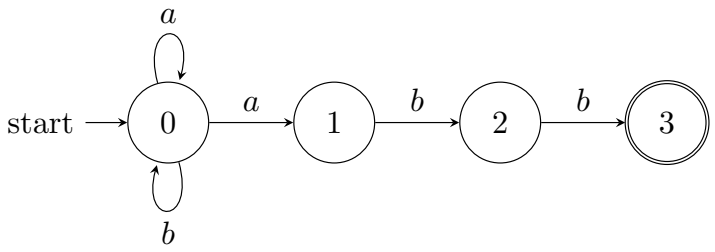


Transition Table:

State	a	b
0	1	0
1	1	2
2	1	3
3	1	0

Simulating FSAs

Representating transition function using transition tables

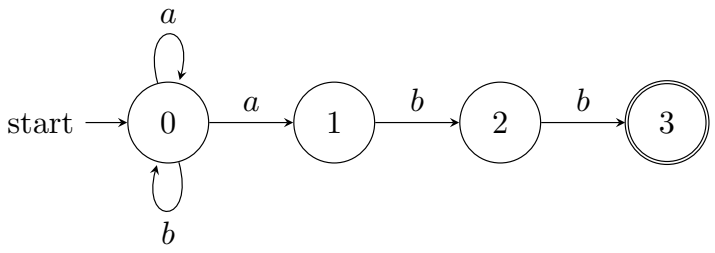


Transition Table:

State	a	b
0		
1		
2		
3		

Simulating FSAs

Representating transition function using transition tables



Transition Table:

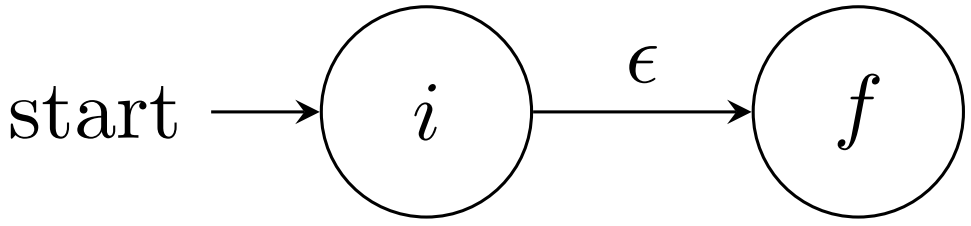
State	a	b
0	{0, 1}	{0}
1	{}	{2}
2	{}	{3}
3	{}	{}

NFA from Regular Expressions

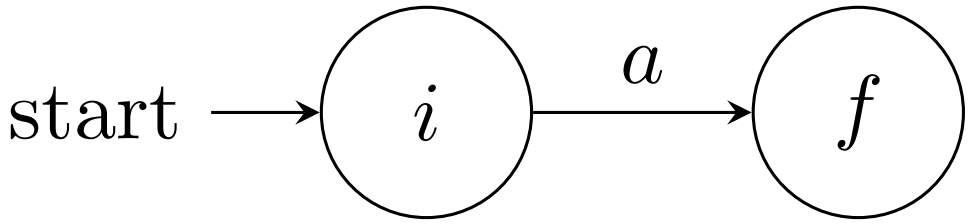
McNaughton-Yamada-Thompson Algorithm

Basis

ϵ



$a \in \sum$



Simulation of DFA

procedure SIMDFA(D, inp)

Simulation of DFA

```
procedure simDFA(D, inp)
    s ← D.s0
    while there is input left do
        c ← nextChar
        s ← D.move(s, c)
        if s = nil then
            break
        end if
    end while
    if s ∈ D.F then
        return true
    else
        return false
    end if
end procedure
```

Simulation of NFA

procedure SIMNFA(N, inp)

Simulation of NFA

procedure SIMNFA(N, inp)
    $S \leftarrow \epsilon$-CLOSURE($N.s_0$)
    $c \leftarrow$ NEXTCHAR
    while there is input left **do**
        $T' \leftarrow$ MOVE(S, c)
        $S \leftarrow \epsilon$-CLOSURE(T')
        $c \leftarrow$ NEXTCHAR
    end while
    if $S \cap N.F \neq \{\}$ **then**
        return true
    else
        return false
    end if
end procedure

Computing ϵ-closure

procedure ϵ-CLOSURE(s)

Computing ϵ-closure

procedure ϵ-CLOSURE(s)
    $stack \leftarrow$ COPY(s)
    $ep \leftarrow$ COPY(s)
    while $stack$ is not empty **do**
        $t \leftarrow stack$.POP
        $U \leftarrow \{u : u \in M.Trans[t, \epsilon]\}$
        for $u \in U$ **do**
            if $u \notin ep$ **then**
                ep.ADD(u)
                $stack$.PUSH(u)
            end if
        end for
    end while
    return ep
end procedure

NFA to DFA

procedure NFA2DFA(N)

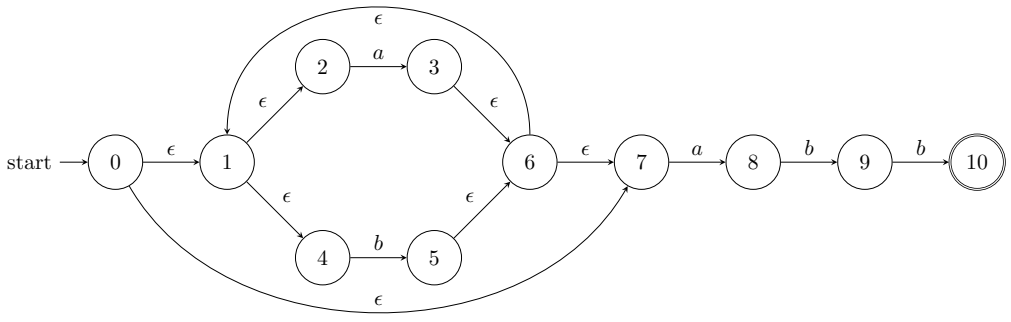
NFA to DFA

procedure NFA2DFA(N)
    $s'_0 \leftarrow \epsilon$-CLOSURE($N.s_0$)
    add s'_0 to $D.states$
    UNMARK(s'_0)
    while there is unmarked state T in $D.states$ **do**
        MARK(T)
        for all $a \in \sum$ **do**
            $T' \leftarrow$ MOVE(T, a)
            $\mathcal{U} \leftarrow \epsilon$-CLOSURE($T'$)
            if $\mathcal{U} \notin D.states$ **then**
                add $\mathcal{U}$ to $D.states$
                UNMARK($\mathcal{U}$)
            end if
        end for
    end while
    return D
end procedure

NFA to DFA

Example

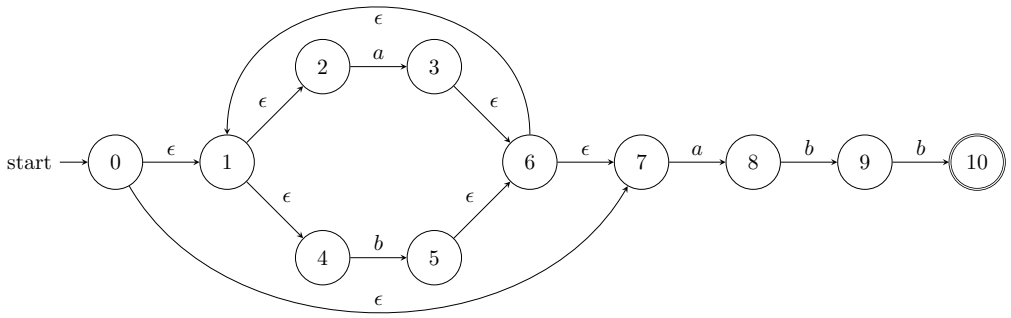
NFA:



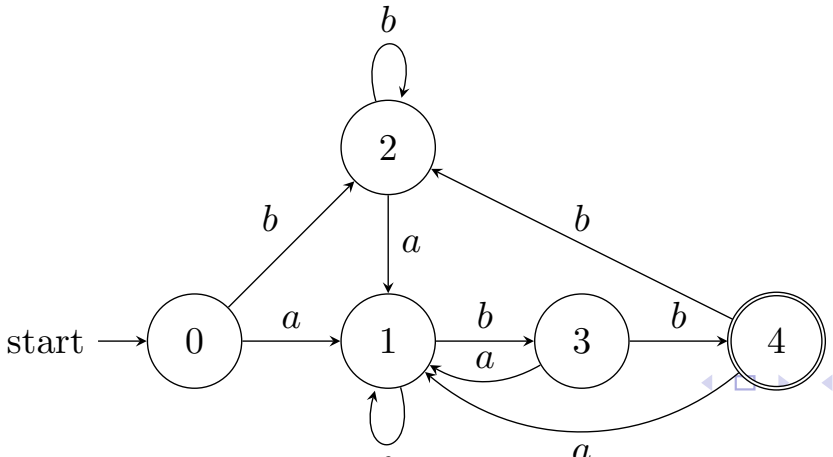
NFA to DFA

Example

NFA:

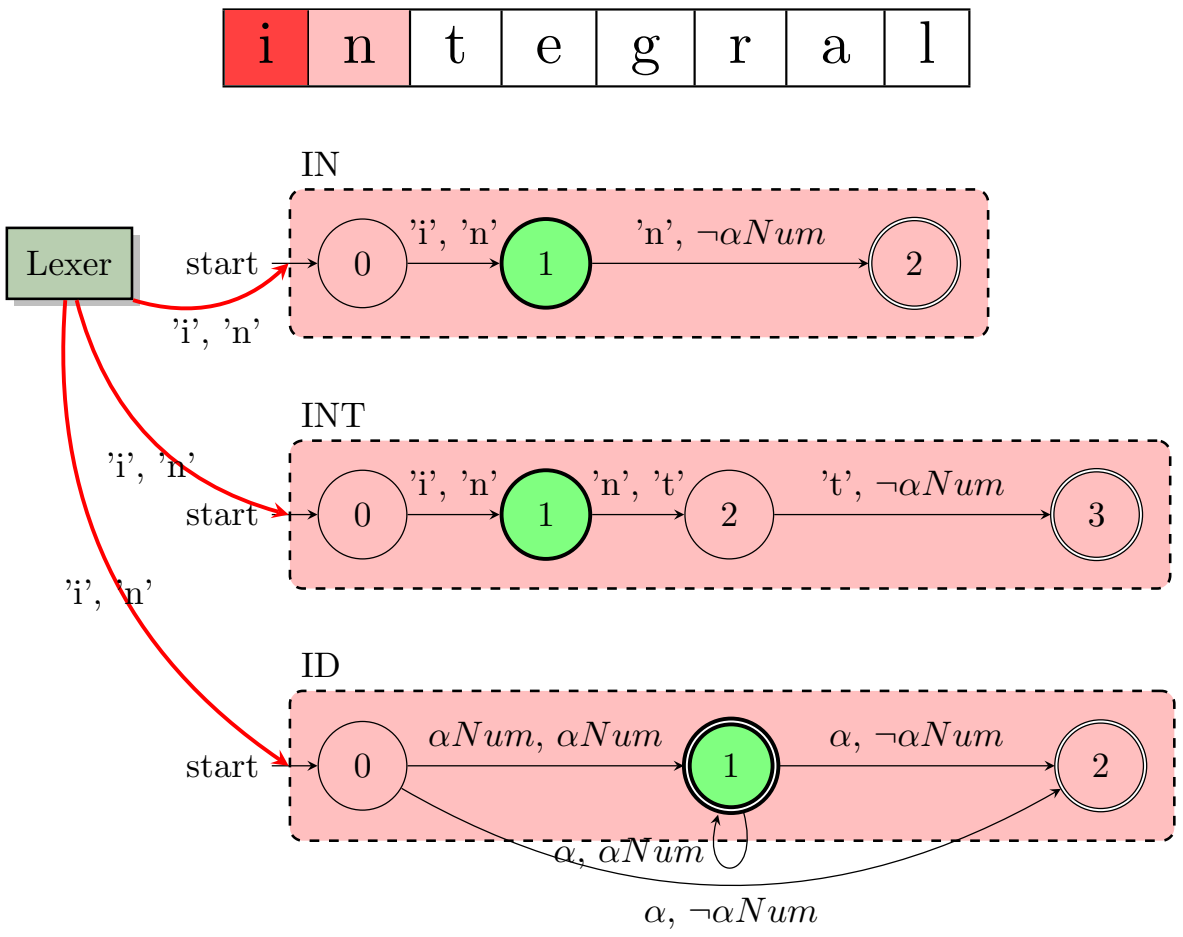


DFA:



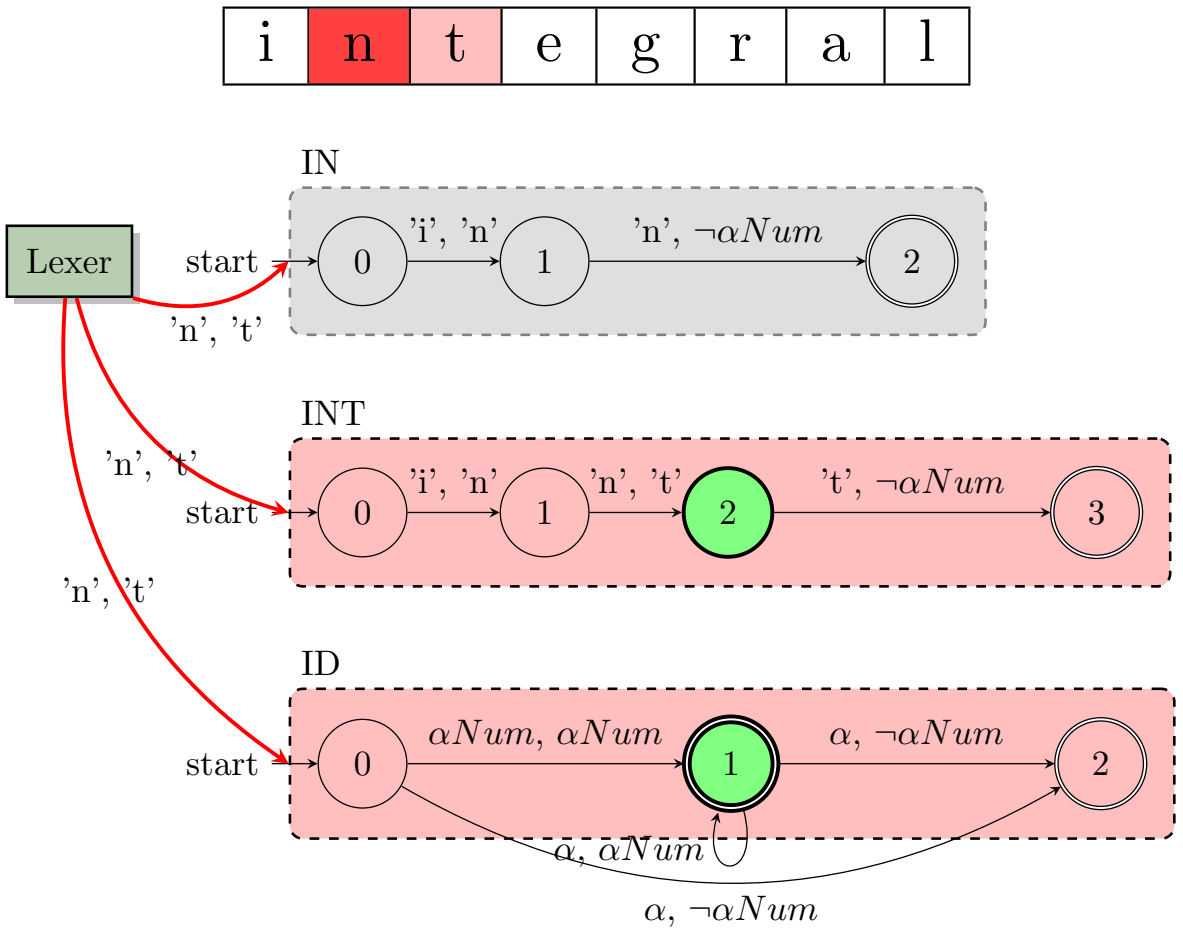
Implementation of Lexical Analysis

Manual Implementation of Lexical Analysers – Example



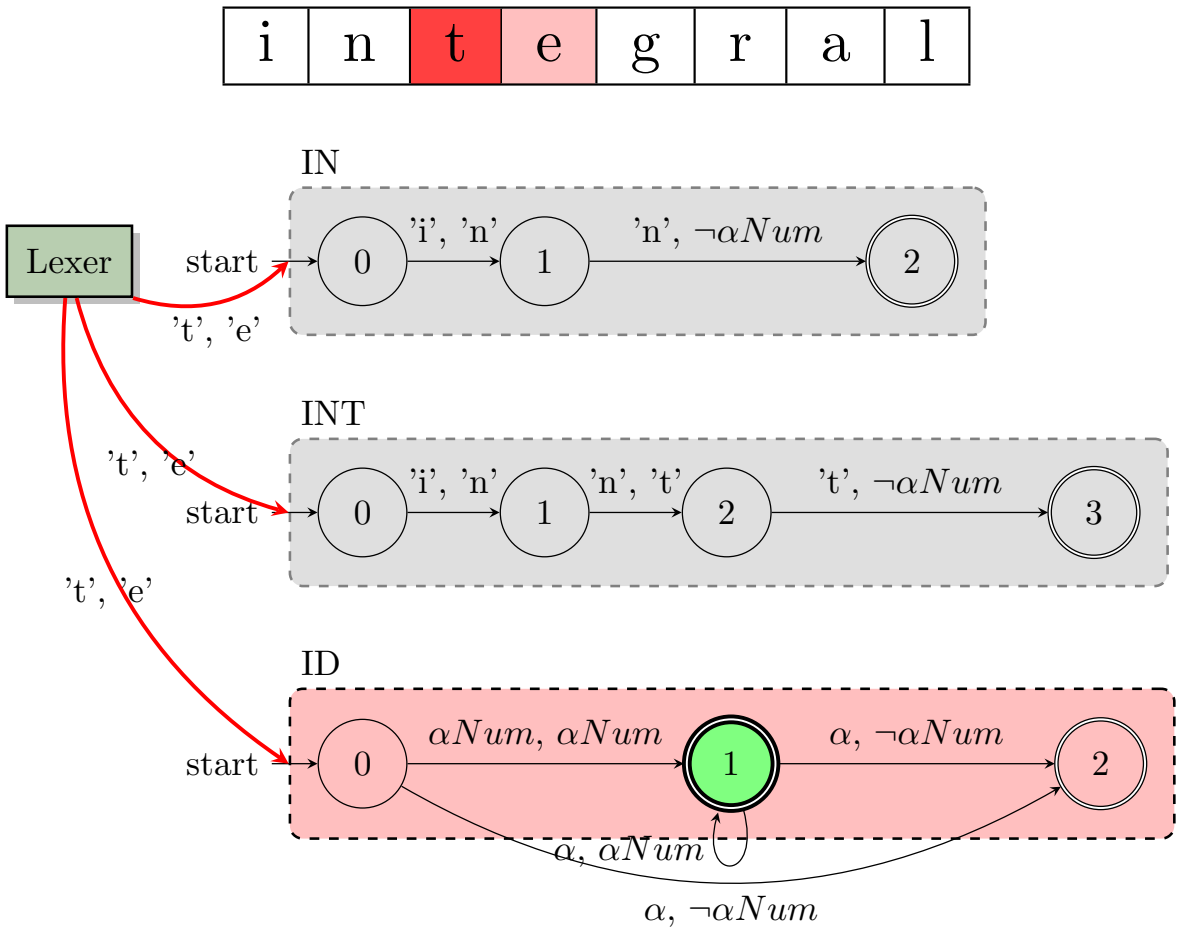
Implementation of Lexical Analysis

Manual Implementation of Lexical Analysers – Example



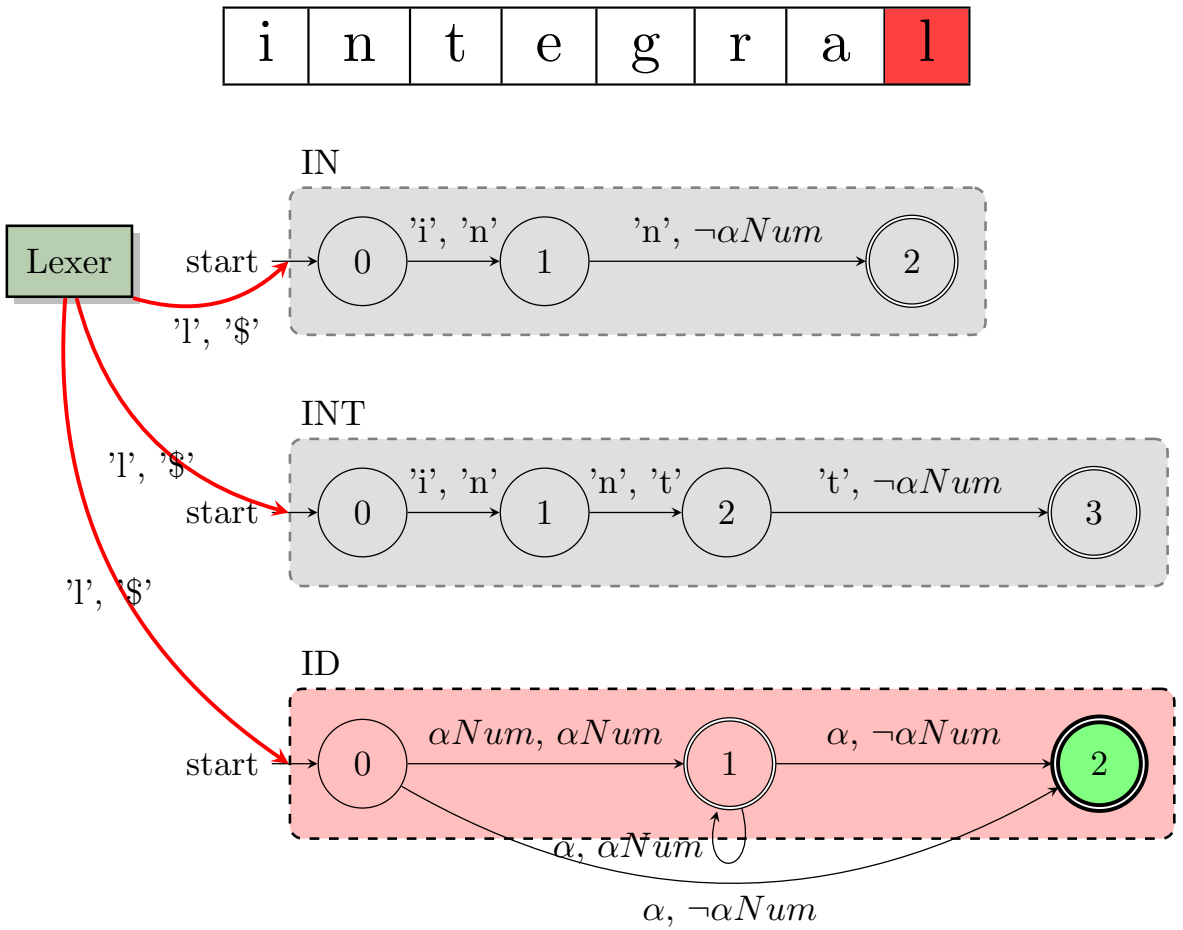
Implementation of Lexical Analysis

Manual Implementation of Lexical Analysers – Example



Implementation of Lexical Analysis

Manual Implementation of Lexical Analysers – Example



Implementation of Lexical Analysis

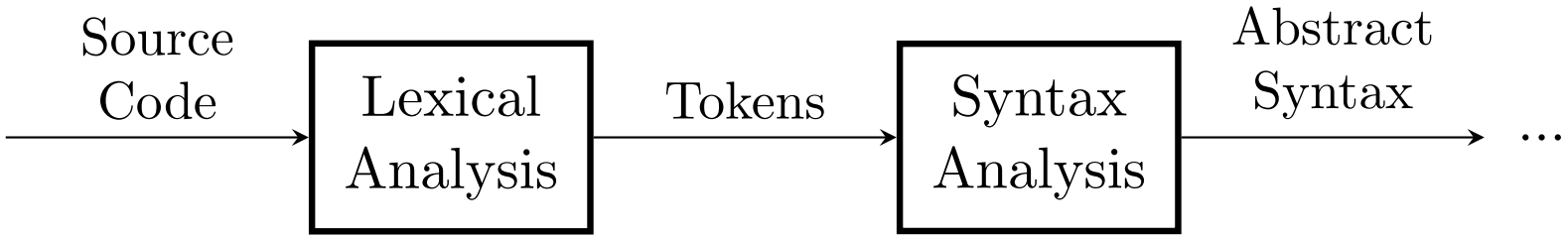
Manual Implementation of Lexical Analysers

Important Points

1. The scanners must be ordered carefully.
2. Alternative design.
   - No separate scanner for keywords
   - Maintain a table of keywords
   - On detecting an identifier, check if it's a keyword. If yes, return suitable token. Otherwise, return identifier.

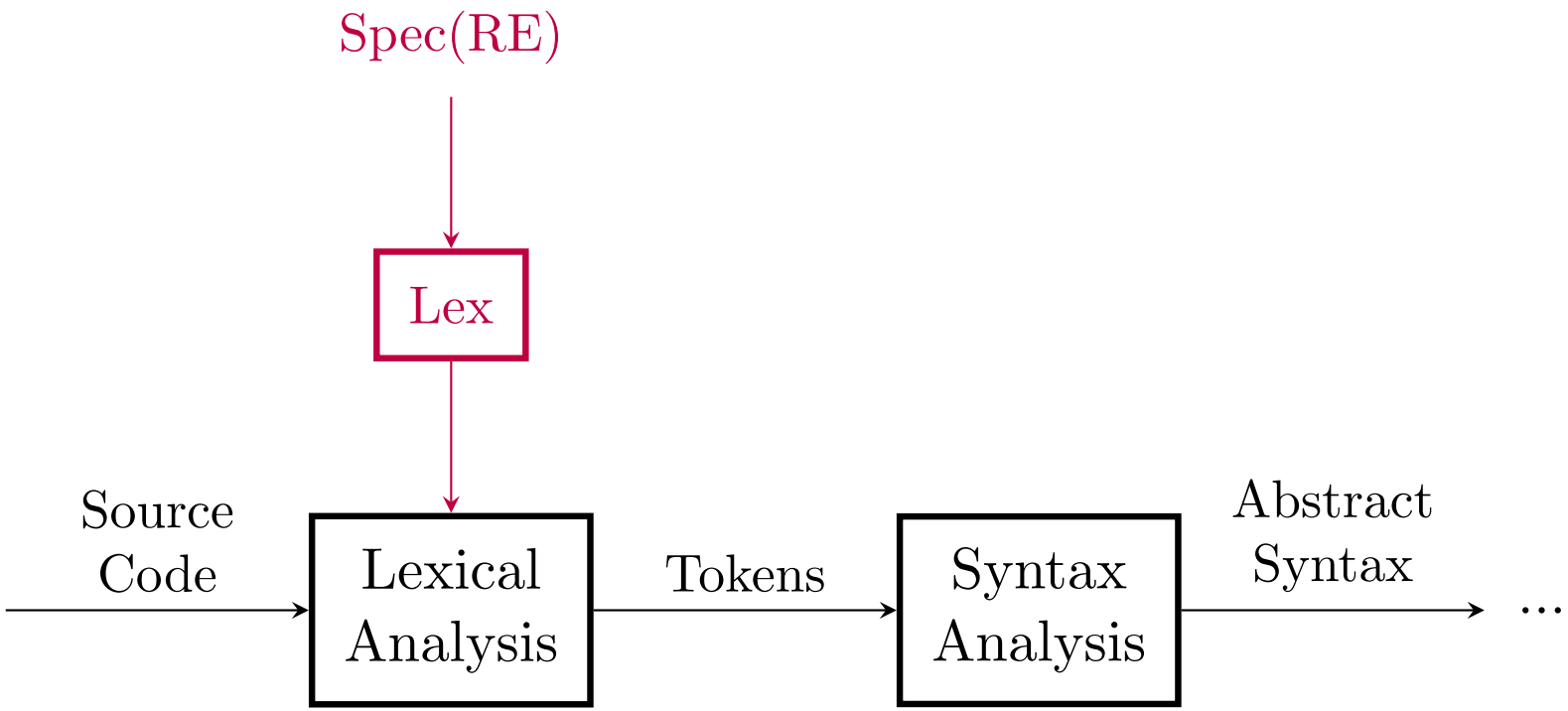
Implementation of Lexical Analysis

Automatic Generation of Lexical Analysers



Implementation of Lexical Analysis

Automatic Generation of Lexical Analysers



Implementation of Lexical Analysis

Automatic Generation of Lexical Analysers