Motivating Example

• Token set:
  – assign -> :=
  – plus -> +
  – minus -> -
  – times -> *
  – div -> /
  – lparen -> (  
  – rparen -> )
  – id -> letter(letter|digit)*
  – number -> digit digit*|digit*(digit|digit.|digit)*

Motivating Example

• What are the lexemes in the string “a_var := b * 3″ ?
• What are the corresponding tokens ?
• How do you identify the tokens?
A naïve scanner algorithm

1. read characters one at a time (e.g., cur_char) with one look-ahead character lh_char
2. if cur_char is one of the one-character tokens { ( ) [ ] < > , ; = + - } then return the token
3. if cur_char is ';', check lh_char
   if lh_char is '=', then return token ":="
   else report error
3. if cur_char is a digit, then read any additional digits with at most one '.' and return number
   else report error
4. if cur_char is a letter, then read any additional letters and digits and return id
   ...

Pictorial representation of the scanner as a finite automaton

- This is a deterministic finite automaton (DFA)
Can we automate the scanner implementation?

• Yes. A scanner generator tool can generate the implementation from regular expressions with four steps
  – Convert regular expressions to Nondeterministic Finite Automata (NFA)
  – Convert NFA to Deterministic Finite Automata (DFA)
  – Minimize the DFA
  – Implement DFA as switch statements

Finite Automaton (FA)

• A simple idealized machine to recognize patterns in character sequences
• Its job is to accept or reject an input depending on whether the pattern defined by the FA occurs in the input
Revisit the Example

• Can an FA $M$ accept a string $w$?
  – The machine starts in the start state
  – Given each character of $w$, the machine will transition from state to state according to predefined transition rules
  – $M$ accepts $w$ if the last input of $w$ causes the machine to halt properly. Otherwise, it is said that $M$ rejects $w$.

Finite Automaton & Regular Expression

• They have equivalent expressive power
  – For every regular expression RE, there is a corresponding FA that accepts the set of strings generated by RE
  – For every FA, there is a corresponding RE that generates the set of strings accepted by FA
Two types of FA

- **Deterministic Finite Automaton (DFA)**
  - Each transition is uniquely determined by its source state and input symbol
  - Reading an input symbol is required for each state transition
- **Nondeterministic Finite Automaton (NFA)**
  - For some state and input symbol, the next state may be one or more possible states
  - Epsilon transitions: arrows labeled by the empty string symbol \( \varepsilon \)

---

**Deterministic Finite Automaton (DFA)**

- 5-tuple, \((Q, \Sigma, \delta, q_0, F)\), consisting of
  - a finite set of states \((Q)\)
  - a finite set of input symbols called the alphabet \((\Sigma)\)
  - a transition function \((\delta : Q \times \Sigma \rightarrow Q)\)
  - an initial or start state \((q_0 \in Q)\)
  - a set of accept states \((F \subseteq Q)\)
Deterministic Finite Automaton (DFA)

Let $w = a_1a_2 \ldots a_n$ be a string over the alphabet $\Sigma$. The automaton $M$ accepts the string $w$ if a sequence of states, $r_0, r_1, \ldots, r_n$, exists in $Q$ with the following conditions:

- $r_0 = q_0$
- $r_{i+1} = \delta(r_i, a_{i+1})$, for $i = 0, \ldots, n-1$
- $r_n \in F$.

DFA Example

$M = (Q, \Sigma, \delta, q_0, F)$ where

- $Q = \{S_1, S_2\}$,
- $\Sigma = \{0, 1\}$,
- $q_0 = S_1$,
- $F = \{S_1\}$, and
- $\delta$ is defined by the following state transition table:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_1$</td>
<td>$S_2$</td>
<td>$S_1$</td>
</tr>
<tr>
<td>$S_2$</td>
<td>$S_1$</td>
<td>$S_2$</td>
</tr>
</tbody>
</table>

State transition diagram
What is the transition diagram of the following transition table? [2]

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\rightarrow q_0)</td>
<td>(q_2)</td>
<td>(q_0)</td>
</tr>
<tr>
<td>*(q_1)</td>
<td>(q_1)</td>
<td>(q_1)</td>
</tr>
<tr>
<td>(q_2)</td>
<td>(q_2)</td>
<td>(q_1)</td>
</tr>
</tbody>
</table>

What is the transition table of the following transition diagram?

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\rightarrow A)</td>
<td>(C)</td>
<td>(B)</td>
</tr>
<tr>
<td>(B)</td>
<td>(C)</td>
<td>(B)</td>
</tr>
<tr>
<td>(C)</td>
<td>(C)</td>
<td>(D)</td>
</tr>
<tr>
<td>*(D)</td>
<td>(D)</td>
<td>(D)</td>
</tr>
</tbody>
</table>
Nondeterministic Finite Automaton (NFA)

• Let $P(Q)$ denote the power set of $Q$

• 5-tuple, $(Q, \Sigma, \Delta, q_0, F)$, consisting of
  – a finite set of states $Q$
  – a finite set of input symbols $\Sigma$
  – a transition function $\Delta : Q \times \Sigma \rightarrow P(Q)$
  – an initial (or start) state $q_0 \in Q$
  – a set of accept states ($F \subseteq Q$)

Nondeterministic Finite Automaton (NFA)

• Let $w = a_1 a_2 ... a_n$ be a word over the alphabet $\Sigma$. The automaton $M$ accepts the word $w$ if a sequence of states, $r_0, r_1, ..., r_n$, exists in $Q$ with the following conditions:
  – $r_0 = q_0$
  – $r_{i+1} \in \Delta(r_i, a_{i+1})$, for $i = 0, ..., n-1$
  – $r_n \in F$. 
An NFA Example [3]

<table>
<thead>
<tr>
<th>State</th>
<th>0</th>
<th>1</th>
<th>(\varepsilon)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(S_0)</td>
<td>{}</td>
<td>{}</td>
<td>{(S_1, S_3)}</td>
</tr>
<tr>
<td>(S_1)</td>
<td>({S_2})</td>
<td>({S_1})</td>
<td>{}</td>
</tr>
<tr>
<td>(S_2)</td>
<td>({S_1})</td>
<td>({S_2})</td>
<td>{}</td>
</tr>
<tr>
<td>(S_3)</td>
<td>({S_3})</td>
<td>({S_4})</td>
<td>{}</td>
</tr>
<tr>
<td>(S_4)</td>
<td>({S_4})</td>
<td>({S_3})</td>
<td>{}</td>
</tr>
</tbody>
</table>

From a Regular Expression to an NFA

a) Base case
b) Concatenation
c) Alternation
d) Kleene closure

Note: Build NFA in a top-down manner by breaking components level-by-level
An Example

- \((a|b)^*abb\)
  - Step 1:
    - \((a|b)^*\)
      - Start
      - \(a\) \(b\) \(b\)
  - Step 2:
    - \(\epsilon\) \(a|b\) \(\epsilon\)
      - Start
      - \(a\) \(b\) \(b\)
  - Step 3:
    - \(\epsilon\) \(\epsilon\) \(a\) \(b\) \(b\) \(\epsilon\)
      - Start
      - \(a\) \(b\) \(b\) \(\epsilon\) \(\epsilon\) \(\epsilon\) \(\epsilon\) \(\epsilon\) \(\epsilon\)

From an NFA to a DFA

- Insight
  - Remove ambiguous transition by merging states
  - Each DFA state corresponds to a set of equivalent NFA states
  - The NFA states within an equivalent set have \(\epsilon\)-transitions among themselves, or they are reachable via the same input symbol from states in another equivalent set
From an NFA to a DFA

Define $\varepsilon$-closure(s) as the set of NFA states reachable from state $s$ on $\varepsilon$-transitions alone.

Algorithm:
- push $\varepsilon$-closure(start) on stack
- while stack is not empty do
  - pop an element $T$
  - mark $T$ as processed
  - for each input symbol $a$ do
    - find all states reachable from any state in $T$ via $a$,
    - also find the $\varepsilon$-closure of those newly found states,
    - if the set is already processed
      - get the assigned label $U$
    - else
      - assign a new label to the state set, such as $U$
      - put $U$ on stack
    - Trans[$T, a$] := $U$, where $T$ and $U$ are new states in DFA, and their transition is labeled with $a$