Ignore § 2.4.4 in EaC2e.
Read, instead, the excerpt from EaC3e (on “Lectures” page).

Lexical Analysis, III

Comp 412

Updated after lecture
The Plan for Scanner Construction

**RE → NFA (Thompson’s construction)**
- Build an NFA for each term in the RE
- Combine them in patterns that model the operators

**NFA → DFA (Subset construction)**
- Build a DFA that simulates the NFA

**DFA → Minimal DFA**
- Hopcroft’s algorithm
- Brzozowski’s algorithm

**Minimal DFA → Scanner**
- See § 2.5 in EaC2e

**DFA → RE**
- All pairs, all paths problem
- Union together paths from $s_0$ to a final state

**The Cycle of Constructions**
DFA Minimization (Hopcroft’s Algorithm)

The Big Picture
- Discover sets of behaviorally equivalent states in the NFA
- Represent each such set with a single new state in the DFA

Two states \( s_i \) and \( s_j \) are **behaviorally equivalent if and only if**:
- \( \forall c \in \Sigma, \) transitions from \( s_i \) & \( s_j \) on \( c \) lead to equivalent states
- The set of paths leading from \( s_i \) & \( s_j \) are equivalent

A **partition** \( P \) of a set \( S \):
- A collection of subsets of \( P \) such that each state \( s \) is in exactly one \( p_i \in P \)

The algorithm iteratively constructs partitions of the NFA’s set of states

We want a partition \( P = \{ p_0, p_1, p_2, \ldots, p_n \} \) of \( D \) that has two properties:
1. If \( d_i \) & \( d_j \in p_s \) and \( c \) takes \( d_i \rightarrow d_x \) and \( d_j \rightarrow d_y \), then \( d_x \) & \( d_y \in p_t \), \( \forall c, i, j, s, t \)
2. If \( d_i \) & \( d_j \in p_s \) and \( d_i \in F \) then \( d_j \in F \)

\( D \) is the set of states for the DFA: \( (D, \Sigma, \delta, s_0, D_A) \)
DFA Minimization

Details of the algorithm

• Group states into maximally-sized initial sets, \textit{optimistically} \hspace{1cm} \text{(property 2)}
• Iteratively subdivide those sets, based on transition graph \hspace{1cm} \text{(property 1)}
• States that remain grouped together are equivalent

Initial partition: \( P_0 \) has two sets: \( \{D_A\} \) \& \( \{D - D_A\} \)

DFA Minimization

Property 1 provides the basis for refining, or splitting, the sets

• Assume \( s_i \) \& \( s_j \in p_s \), and \( \delta^{-1}(s_i,a) = s_x \), \& \( \delta^{-1}(s_j,a) = s_y \)
• If \( s_x \) \& \( s_y \) are not in the same set \( p_t \), then \( p_s \) must be split
  – \text{COROLLARY:} \( s_i \) has transition on \( a \), \( s_j \) does not \( \Rightarrow a \) splits \( p_s \)
• A single state in a DFA cannot have two transitions on \( a \)
  – Each \( p_s \) will become a DFA state

Algorithm actually works backward; it looks at what transitions enter \( p \) on character \( c \), and uses that to split the partition \( q \) where those edges begin.
DFA Minimization Algorithm (Worklist version)

\[
\text{Worklist} \leftarrow \{ D_A, \{ D - D_A \} \}
\]

\[
\text{Partition} \leftarrow \{ D_A, \{ D - D_A \} \}
\]

\[\text{While (Worklist} \neq \emptyset) \text{ do}\]

\[\text{select a set } S \text{ from Worklist and remove it}\]

\[\text{for each } \alpha \in \Sigma \text{ do}\]

\[\text{Image} \leftarrow \{ x \mid \delta(x, \alpha) \in S \}\]

\[\text{for each } q \in \text{Partition that has a state in Image do}\]

\[q_1 \leftarrow q \cap \text{Image}\]

\[q_2 \leftarrow q - q_1\]

\[\text{if } q_2 \neq \emptyset \text{ then}\]

\[\text{remove } q \text{ from Partition}\]

\[\text{Partition} \leftarrow \text{Partition} \cup q_1 \cup q_2\]

\[\text{if } q \in \text{Worklist then}\]

\[\text{remove } q \text{ from Worklist}\]

\[\text{Worklist} \leftarrow \text{Worklist} \cup q_1 \cup q_2\]

\[\text{else if } |q_1| \leq |q_2| \text{ then}\]

\[\text{Worklist} \leftarrow \text{Worklist} \cup q_1\]

\[\text{else Worklist} \leftarrow \text{Worklist} \cup q_2\]

\[\text{if } s = q \text{ then}\]

\[\text{break;} \quad // \text{cannot keep working on } s\]

Image is the set of states that have a transition into \(S\) on \(\alpha\):
\[\delta^{-1}(S, \alpha)\]

“split \(q\)”

\(q_1\) is the subset of \(q\) that transitions to \(S\) on \(\alpha\)

\(q_2\) is the rest of \(q\)

And, as an implementation nit, if we just split \(S\) — that is, \(S\) was \(q\) & it split — we need a new \(S\)
Key Idea: Splitting Q Around Transitions on $\alpha$

Partitioning $Q$ around $S$

Assume that $Q$, $R$, $S$, & $T$ are sets in the current approximation to the final partition $Q$ has transitions on $\alpha$ to $R$, $S$, & $T$, so it must split around $\alpha$.

As the algorithm considers $S$ and $\alpha$, it will split $Q$. 
Key Idea: Splitting $Q$ around $S$ and $\alpha$

Find maximal subset of $Q$ that has an $\alpha$-transition into $S$ ($P_1$)

Think of $Q_1$ as the image of $S$ into $Q$ under the inverse of the transition function:

$$Q_1 \leftarrow \delta^{-1}(S, \alpha) \cap Q$$

$Q_2$ must have an $\alpha$-transition to one or more other states in one or more other partitions (e.g., $R$ & $S$), or states with no $\alpha$-transitions.

Otherwise, $Q$ does not split!
DFA Minimization Algorithm (Worklist version)

Worklist $\leftarrow \{ D_A, \{ D - D_A \} \}$
Partition $\leftarrow \{ D_A, \{ D - D_A \} \}$

While (Worklist $\neq \emptyset$) do
    select a set $S$ from Worklist and remove it
    for each $\alpha \in \Sigma$ do
        $\text{Image} \leftarrow \{ x \mid \delta(x, \alpha) \in S \}$
        for each $q \in \text{Partition}$ that has a state in Image do
            $q_1 \leftarrow q \cap \text{Image}$
            $q_2 \leftarrow q - q_1$
            if $q_2 \neq \emptyset$ then
                remove $q$ from Partition
                Partition $\leftarrow$ Partition $\cup$ $q_1 \cup q_2$
            end if
        end for
    end for
    if $q \in \text{Worklist}$ then
        remove $q$ from Worklist
        Worklist $\leftarrow$ Worklist $\cup$ $q_1 \cup q_2$
    else if $|q_1| \leq |q_2|$ then
        Worklist $\leftarrow$ Worklist $\cup$ $q_1$
    else Worklist $\leftarrow$ Worklist $\cup$ $q_2$
    end if
end while
if $s = q$ then
    break; // cannot keep working on $s$
end if

Projection is the set of states that have a transition into $S$ on $\alpha$: $\delta^{-1}(S,\alpha)$
$q_1$ is the subset of $q$ that transitions to $S$ on $\alpha$
$q_2$ is the rest of $q$

And, as an implementation nit, if we just split $S$ — that is, $S$ was $q$ & it split — we need a new $S$
DFA Minimization Algorithm (Worklist version)

\[
\begin{align*}
\text{Worklist} & \leftarrow \{ D_A, \{ D - D_A \} \} \\
\text{Partition} & \leftarrow \{ D_A, \{ D - D_A \} \} \\
\text{While } (\text{Worklist} \neq \emptyset) \text{ do} & \\
& \quad \text{select a set } S \text{ from Worklist and remove it} \\
& \quad \text{for each } \alpha \in \Sigma \text{ do} \\
& \quad \quad \text{Image} \leftarrow \{ x \mid \delta(x, \alpha) \in S \} \\
& \quad \quad \text{for each } q \in \text{Partition that has a state in Image do} \\
& \quad \quad \quad q_1 \leftarrow q \cap \text{Image} \\
& \quad \quad \quad q_2 \leftarrow q - q_1 \\
& \quad \quad \quad \text{if } q_2 \neq \emptyset \text{ then} \\
& \quad \quad \quad \quad \text{remove } q \text{ from Partition} \\
& \quad \quad \quad \quad \text{Partition} \leftarrow \text{Partition} \cup q_1 \cup q_2 \\
& \quad \quad \quad \text{if } q \in \text{Worklist then} \\
& \quad \quad \quad \quad \text{remove } q \text{ from Worklist} \\
& \quad \quad \quad \quad \text{Worklist} \leftarrow \text{Worklist} \cup q_1 \cup q_2 \\
& \quad \quad \quad \text{else if } |q_1| \leq |q_2| \\
& \quad \quad \quad \quad \text{then } \text{Worklist} \leftarrow \text{Worklist} \cup q_1 \\
& \quad \quad \quad \quad \text{else } \text{Worklist} \leftarrow \text{Worklist} \cup q_2 \\
& \quad \quad \quad \text{if } s = q \text{ then} \\
& \quad \quad \quad \quad \text{break; } // \text{ cannot keep working on } s
\end{align*}
\]

One last hack ...

If q is a singleton, we can skip the body of the loop because a singleton cannot split.
### A Detailed Example

<table>
<thead>
<tr>
<th>Step</th>
<th>Partition</th>
<th>W’list</th>
<th>s</th>
<th>c</th>
<th>Image</th>
<th>q</th>
<th>q_1</th>
<th>q_2</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( p_0: {3,5}, p_1: {0,1,2,4} )</td>
<td>( p_0, p_1 )</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
</tbody>
</table>
A Detailed Example

<table>
<thead>
<tr>
<th>Step</th>
<th>Partition</th>
<th>W’list</th>
<th>s</th>
<th>c</th>
<th>Image</th>
<th>q</th>
<th>q₁</th>
<th>q₂</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$p₀: {3,5}, p₁: {0,1,2,4}$</td>
<td>$p₀, p₁$</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>1</td>
<td>$p₀: {3,5}, p₁: {0,1,2,4}$</td>
<td>$p₁$</td>
<td>$p₀$</td>
<td>$e$</td>
<td>$s₂, s₄$</td>
<td>$p₁$</td>
<td>$s₂, s₄$</td>
<td>$s₀, s₁$</td>
<td>$split \ p₁$</td>
</tr>
</tbody>
</table>
## A Detailed Example

<table>
<thead>
<tr>
<th>Step</th>
<th>Partition</th>
<th>W’list</th>
<th>s</th>
<th>c</th>
<th>Image</th>
<th>q</th>
<th>q₁</th>
<th>q₂</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(p₀: {3,5}, p₁: {0,1,2,4})</td>
<td>(p₀, p₁)</td>
<td>—</td>
<td></td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>1</td>
<td>(p₀: {3,5}, p₁: {0,1,2,4})</td>
<td>(p₁)</td>
<td>(p₀)</td>
<td>e</td>
<td>(s₂, s₄)</td>
<td>(p₁)</td>
<td>(s₂, s₄)</td>
<td>(s₀, s₁)</td>
<td>split (p₁)</td>
</tr>
<tr>
<td></td>
<td>(p₀: {3,5}, p₂: {2,4}, p₃: {0,1})</td>
<td>(p₂, p₃)</td>
<td>(p₀)</td>
<td>(f)</td>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>(p₀: {3,5}, p₂: {2,4}, p₃: {0,1})</td>
<td>(p₂, p₃)</td>
<td>(p₀)</td>
<td>(i)</td>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
<td>none</td>
</tr>
</tbody>
</table>
### A Detailed Example

<table>
<thead>
<tr>
<th>Step</th>
<th>Partition</th>
<th>W’list</th>
<th>s</th>
<th>c</th>
<th>Image</th>
<th>q</th>
<th>q₁</th>
<th>q₂</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$p_0: {3,5}$, $p_1: {0,1,2,4}$</td>
<td>$p_0, p_1$</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>1</td>
<td>$p_0: {3,5}$, $p_1: {0,1,2,4}$</td>
<td>$p_1$</td>
<td>$p_0$</td>
<td>$e$</td>
<td>$s_2, s_4$</td>
<td>$p_1$</td>
<td>$s_2, s_4$</td>
<td>$s_0, s_1$</td>
<td>split $p_1$</td>
</tr>
<tr>
<td></td>
<td>$p_0: {3,5}$, $p_2: {2,4}$, $p_3: {0,1}$</td>
<td>$p_2, p_3$</td>
<td>$p_0$</td>
<td>$f$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>$p_0: {3,5}$, $p_2: {2,4}$, $p_3: {0,1}$</td>
<td>$p_2, p_3$</td>
<td>$p_0$</td>
<td>$i$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>none</td>
</tr>
<tr>
<td>2</td>
<td>$p_0: {3,5}$, $p_2: {2,4}$, $p_3: {0,1}$</td>
<td>$p_3$</td>
<td>$p_2$</td>
<td>$e$</td>
<td>$s_1$</td>
<td>$p_3$</td>
<td>$s_1$</td>
<td>$s_0$</td>
<td>split $p_3$</td>
</tr>
</tbody>
</table>
A Detailed Example

<table>
<thead>
<tr>
<th>Step</th>
<th>Partition</th>
<th>W’list</th>
<th>s</th>
<th>c</th>
<th>Image</th>
<th>q</th>
<th>q₁</th>
<th>q₂</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( p₀: {3,5}, p₁: {0,1,2,4} )</td>
<td>( p₀, p₁ )</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>( p₀: {3,5}, p₁: {0,1,2,4} )</td>
<td>( p₁ )</td>
<td>( p₀ )</td>
<td>e</td>
<td>( s₂, s₄ )</td>
<td>( p₁ )</td>
<td>( s₂, s₄ )</td>
<td>( s₀, s₁ )</td>
<td>split ( p₁ )</td>
</tr>
<tr>
<td></td>
<td>( p₀: {3,5}, p₂: {2,4}, p₃: {0,1} )</td>
<td>( p₂, p₃ )</td>
<td>( p₀ )</td>
<td>f</td>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>( p₂, p₃ )</td>
<td>( p₀ )</td>
<td>i</td>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
<td>none</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>( p₀: {3,5}, p₂: {2,4}, p₃: {0,1} )</td>
<td>( p₃ )</td>
<td>( p₂ )</td>
<td>e</td>
<td>( s₁ )</td>
<td>( p₃ )</td>
<td>( s₁ )</td>
<td>( s₀ )</td>
<td>split ( p₃ )</td>
</tr>
<tr>
<td></td>
<td>( p₀: {3,5}, p₂: {2,4}, p₄: {1}, p₅: {0} )</td>
<td>( p₄, p₅ )</td>
<td>( p₂ )</td>
<td>f</td>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>( p₄, p₅ )</td>
<td>( p₂ )</td>
<td>i</td>
<td>( s₁ )</td>
<td>( p₄ )</td>
<td>( s₁ )</td>
<td>( \emptyset )</td>
<td>none</td>
<td></td>
</tr>
</tbody>
</table>
## A Detailed Example

<table>
<thead>
<tr>
<th>Step</th>
<th>Partition</th>
<th>W’list</th>
<th>s</th>
<th>c</th>
<th>Image</th>
<th>q</th>
<th>q₁</th>
<th>q₂</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$p_0: {3,5}, p_1: {0,1,2,4}$</td>
<td>$p_0, p_1$</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>1</td>
<td>$p_0: {3,5}, p_1: {0,1,2,4}$</td>
<td>$p_1$</td>
<td>$p_0$</td>
<td>$e$</td>
<td>$s_2, s_4$</td>
<td>$p_1$</td>
<td>$s_2, s_4$</td>
<td>$s_0, s_1$</td>
<td>split $p_1$</td>
</tr>
<tr>
<td></td>
<td>$p_0: {3,5}, p_2: {2,4}, p_3: {0,1}$</td>
<td>$p_2, p_3$</td>
<td>$p_0$</td>
<td>$f$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>$p_2, p_3$</td>
<td>$p_0$</td>
<td>$i$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>none</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>$p_0: {3,5}, p_2: {2,4}, p_3: {0,1}$</td>
<td>$p_3$</td>
<td>$p_2$</td>
<td>$e$</td>
<td>$s_1$</td>
<td>$p_3$</td>
<td>$s_1$</td>
<td>$s_0$</td>
<td>split $p_3$</td>
</tr>
<tr>
<td></td>
<td>$p_0: {3,5}, p_2: {2,4}, p_4: {1}$, $p_5: {0}$</td>
<td>$p_4, p_5$</td>
<td>$p_2$</td>
<td>$f$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>$p_4, p_5$</td>
<td>$p_2$</td>
<td>$i$</td>
<td>$s_1$</td>
<td>$p_4$</td>
<td>$s_1$</td>
<td>$\emptyset$</td>
<td>none</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>$p_5$</td>
<td>$p_4$</td>
<td>$e$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>none</td>
<td></td>
</tr>
</tbody>
</table>
A Detailed Example

<table>
<thead>
<tr>
<th>Step</th>
<th>Partition</th>
<th>W’list</th>
<th>s</th>
<th>c</th>
<th>Image</th>
<th>q</th>
<th>q₁</th>
<th>q₂</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(p_0: {3,5}, p_1: {0,1,2,4})</td>
<td>(p_0, p_1)</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>1</td>
<td>(p_0: {3,5}, p_1: {0,1,2,4})</td>
<td>(p_1)</td>
<td>(p_0)</td>
<td>e</td>
<td>(s_2, s_4)</td>
<td>(p_1)</td>
<td>(s_2, s_4)</td>
<td>(s_0, s_1)</td>
<td>split (p_1)</td>
</tr>
<tr>
<td></td>
<td>(p_0: {3,5}, p_2: {2,4}, p_3: {0,1})</td>
<td>(p_2, p_3)</td>
<td>(p_0)</td>
<td>f</td>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>(p_2, p_3)</td>
<td>(p_0)</td>
<td>i</td>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
<td>none</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>(p_0: {3,5}, p_2: {2,4}, p_3: {0,1})</td>
<td>(p_3)</td>
<td>(p_2)</td>
<td>e</td>
<td>(s_1)</td>
<td>(p_3)</td>
<td>(s_1)</td>
<td>(s_0)</td>
<td>split (p_3)</td>
</tr>
<tr>
<td></td>
<td>(p_0: {3,5}, p_2: {2,4}, p_4: {1}, p_5: {0})</td>
<td>(p_4, p_5)</td>
<td>(p_2)</td>
<td>f</td>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>(p_4, p_5)</td>
<td>(p_2)</td>
<td>i</td>
<td>(s_1)</td>
<td>(p_4)</td>
<td>(s_1)</td>
<td>(\emptyset)</td>
<td>none</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>(p_0: {3,5}, p_2: {2,4}, p_4: {1}, p_5: {0})</td>
<td>(p_5)</td>
<td>(p_4)</td>
<td>e</td>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>(p_5)</td>
<td>(p_4)</td>
<td>f</td>
<td>(s_0)</td>
<td>(p_5)</td>
<td>(s_0)</td>
<td>(\emptyset)</td>
<td>none</td>
<td></td>
</tr>
</tbody>
</table>
A Detailed Example

<table>
<thead>
<tr>
<th>Step</th>
<th>Partition</th>
<th>W’list</th>
<th>s</th>
<th>c</th>
<th>Image</th>
<th>q</th>
<th>q₁</th>
<th>q₂</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(p_0: {3,5}, p_1: {0,1,2,4})</td>
<td>(p_0, p_1)</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>1</td>
<td>(p_0: {3,5}, p_1: {0,1,2,4})</td>
<td>(p_1)</td>
<td>(p_0)</td>
<td>(e)</td>
<td>(s_2, s_4)</td>
<td>(p_1)</td>
<td>(s_2, s_4)</td>
<td>(s_0, s_1)</td>
<td>split (p_1)</td>
</tr>
<tr>
<td></td>
<td>(p_0: {3,5}, p_2: {2,4}, p_3: {0,1})</td>
<td>(p_2, p_3)</td>
<td>(p_0)</td>
<td>(f)</td>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
<td>none</td>
</tr>
<tr>
<td>2</td>
<td>(p_0: {3,5}, p_2: {2,4}, p_3: {0,1})</td>
<td>(p_3)</td>
<td>(p_2)</td>
<td>(e)</td>
<td>(s_1)</td>
<td>(p_3)</td>
<td>(s_1)</td>
<td>(s_0)</td>
<td>split (p_3)</td>
</tr>
<tr>
<td></td>
<td>(p_0: {3,5}, p_2: {2,4}, p_4: {1}, p_5: {0})</td>
<td>(p_4, p_5)</td>
<td>(p_2)</td>
<td>(f)</td>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
<td>none</td>
</tr>
<tr>
<td>3</td>
<td>(p_0: {3,5}, p_2: {2,4}, p_4: {1}, p_5: {0})</td>
<td>(p_5)</td>
<td>(p_4)</td>
<td>(e)</td>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
<td>none</td>
</tr>
</tbody>
</table>
### A Detailed Example

<table>
<thead>
<tr>
<th>Step</th>
<th>Partition</th>
<th>W’list</th>
<th>s</th>
<th>c</th>
<th>Image</th>
<th>q</th>
<th>q&lt;sub&gt;1&lt;/sub&gt;</th>
<th>q&lt;sub&gt;2&lt;/sub&gt;</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( p_0: {3,5}, p_1: {0,1,2,4} )</td>
<td>( p_0, p_1 )</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>1</td>
<td>( p_0: {3,5}, p_1: {0,1,2,4} )</td>
<td>( p_1 )</td>
<td>( p_0 )</td>
<td>e</td>
<td>( s_2, s_4 )</td>
<td>( p_1 )</td>
<td>( s_2, s_4 )</td>
<td>( s_0, s_1 )</td>
<td>\text{split } p_1</td>
</tr>
<tr>
<td></td>
<td>( p_0: {3,5}, p_2: {2,4}, p_3: {0,1} )</td>
<td>( p_2, p_3 )</td>
<td>( p_0 )</td>
<td>f</td>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>( p_2, p_3 )</td>
<td>( p_0 )</td>
<td>i</td>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
<td>none</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>( p_0: {3,5}, p_2: {2,4}, p_3: {0,1} )</td>
<td>( p_3 )</td>
<td>( p_2 )</td>
<td>e</td>
<td>( s_1 )</td>
<td>( p_3 )</td>
<td>( s_1 )</td>
<td>( s_0 )</td>
<td>\text{split } p_3</td>
</tr>
<tr>
<td></td>
<td>( p_0: {3,5}, p_2: {2,4}, p_4: {1}, p_5: {0} )</td>
<td>( p_4, p_5 )</td>
<td>( p_2 )</td>
<td>f</td>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>( p_4, p_5 )</td>
<td>( p_2 )</td>
<td>i</td>
<td>( s_1 )</td>
<td>( p_4 )</td>
<td>( s_1 )</td>
<td>( \emptyset )</td>
<td>none</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>( p_0: {3,5}, p_2: {2,4}, p_4: {1}, p_5: {0} )</td>
<td>( p_5 )</td>
<td>( p_4 )</td>
<td>e</td>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>( p_5 )</td>
<td>( p_4 )</td>
<td>f</td>
<td>( s_0 )</td>
<td>( p_5 )</td>
<td>( s_0 )</td>
<td>( \emptyset )</td>
<td>none</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( p_5 )</td>
<td>( p_4 )</td>
<td>i</td>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
<td>none</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>( p_0: {3,5}, p_2: {2,4}, p_4: {1}, p_5: {0} )</td>
<td>( \emptyset )</td>
<td>( p_5 )</td>
<td>e</td>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>( \emptyset )</td>
<td>( p_5 )</td>
<td>f</td>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
<td>none</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( \emptyset )</td>
<td>( p_5 )</td>
<td>i</td>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
<td>none</td>
<td></td>
</tr>
</tbody>
</table>

**Diagram:**

- **Nodes:** \( s_0, s_1, s_2, s_3, s_4, s_5 \)
- **Edges:**
  - \( e \) from \( s_0 \) to \( s_3 \)
  - \( e \) from \( s_4 \) to \( s_5 \)
  - \( f \) from \( s_0 \) to \( s_1 \)
  - \( i \) from \( s_4 \) to \( s_1 \)
A Detailed Example

<table>
<thead>
<tr>
<th>Step</th>
<th>Partition</th>
<th>W’list</th>
<th>s</th>
<th>c</th>
<th>Image</th>
<th>q</th>
<th>q₁</th>
<th>q₂</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(p₀: {3,5}, p₁: {0,1,2,4})</td>
<td>(p₀, p₁)</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>1</td>
<td>(p₀: {3,5}, p₁: {0,1,2,4})</td>
<td>(p₁)</td>
<td>(p₀)</td>
<td>(e)</td>
<td>(s₂, s₄)</td>
<td>(p₁)</td>
<td>(s₂, s₄)</td>
<td>(s₀, s₁)</td>
<td>split (p₁)</td>
</tr>
<tr>
<td></td>
<td>(p₀: {3,5}, p₂: {2,4}, p₃: {0,1})</td>
<td>(p₀)</td>
<td>(f)</td>
<td>(∅)</td>
<td>(∅)</td>
<td>(∅)</td>
<td>(∅)</td>
<td>none</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(p₀: {3,5}, p₂: {2,4}, p₃: {0,1})</td>
<td>(p₀)</td>
<td>(i)</td>
<td>(∅)</td>
<td>(∅)</td>
<td>(∅)</td>
<td>(∅)</td>
<td>none</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>(p₀: {3,5}, p₂: {2,4}, p₃: {0,1})</td>
<td>(p₃)</td>
<td>(p₂)</td>
<td>(e)</td>
<td>(s₁)</td>
<td>(p₃)</td>
<td>(s₁)</td>
<td>(s₀)</td>
<td>split (p₃)</td>
</tr>
<tr>
<td></td>
<td>(p₀: {3,5}, p₂: {2,4}, p₄: {1}, p₅: {0})</td>
<td>(p₂)</td>
<td>(f)</td>
<td>(∅)</td>
<td>(∅)</td>
<td>(∅)</td>
<td>(∅)</td>
<td>none</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(p₀: {3,5}, p₂: {2,4}, p₄: {1}, p₅: {0})</td>
<td>(p₂)</td>
<td>(i)</td>
<td>(s₁)</td>
<td>(p₄)</td>
<td>(s₁)</td>
<td>(∅)</td>
<td>none</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>(p₀: {3,5}, p₂: {2,4}, p₄: {1}, p₅: {0})</td>
<td>(p₅)</td>
<td>(p₄)</td>
<td>(e)</td>
<td>(∅)</td>
<td>(∅)</td>
<td>(∅)</td>
<td>(∅)</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>(p₀: {3,5}, p₂: {2,4}, p₄: {1}, p₅: {0})</td>
<td>(p₅)</td>
<td>(p₄)</td>
<td>(f)</td>
<td>(s₀)</td>
<td>(p₅)</td>
<td>(s₀)</td>
<td>(∅)</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>none</td>
<td></td>
</tr>
<tr>
<td></td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>none</td>
<td></td>
</tr>
</tbody>
</table>

Reconstructing the DFA

- Each set in Partition forms a state
- For each line in the table where \(q₁ \neq ∅\), add an edge from \(q₁\) to \(s\) labelled \(c\)
A Detailed Example

<table>
<thead>
<tr>
<th>Step</th>
<th>Partition</th>
<th>W’list</th>
<th>s</th>
<th>c</th>
<th>Image</th>
<th>q</th>
<th>q₁</th>
<th>q₂</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(p₀): {3,5}, (p₁): {0,1,2,4}</td>
<td>(p₀, p₁)</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>1</td>
<td>(p₀): {3,5}, (p₁): {0,1,2,4}</td>
<td>(p₁)</td>
<td>(p₀)</td>
<td>e</td>
<td>(s₂, s₄)</td>
<td>(p₁)</td>
<td>(s₂, s₄)</td>
<td>(s₀, s₁)</td>
<td>split (p₁)</td>
</tr>
<tr>
<td></td>
<td>(p₀): {3,5}, (p₂): {2,4}, (p₃): {0,1}</td>
<td>(p₀)</td>
<td>(f)</td>
<td>(∅)</td>
<td>(∅)</td>
<td>(∅)</td>
<td>(∅)</td>
<td>none</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(p₀): {3,5}, (p₂): {2,4}, (p₃): {0,1}</td>
<td>(p₀)</td>
<td>(i)</td>
<td>(∅)</td>
<td>(∅)</td>
<td>(∅)</td>
<td>(∅)</td>
<td>none</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>(p₀): {3,5}, (p₂): {2,4}, (p₃): {0,1}</td>
<td>(p₃)</td>
<td>(p₂)</td>
<td>e</td>
<td>(s₁)</td>
<td>(p₃)</td>
<td>(s₁)</td>
<td>(s₀)</td>
<td>split (p₃)</td>
</tr>
<tr>
<td></td>
<td>(p₀): {3,5}, (p₂): {2,4}, (p₄): {1}, (p₅): {0}</td>
<td>(p₂)</td>
<td>(f)</td>
<td>(∅)</td>
<td>(∅)</td>
<td>(∅)</td>
<td>(∅)</td>
<td>none</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(p₀): {3,5}, (p₂): {2,4}, (p₄): {1}, (p₅): {0}</td>
<td>(p₂)</td>
<td>(i)</td>
<td>(s₁)</td>
<td>(p₄)</td>
<td>(s₁)</td>
<td>(∅)</td>
<td>none</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>(p₀): {3,5}, (p₂): {2,4}, (p₄): {1}, (p₅): {0}</td>
<td>(p₅)</td>
<td>(p₄)</td>
<td>e</td>
<td>(∅)</td>
<td>(∅)</td>
<td>(∅)</td>
<td>(∅)</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>(p₀): {3,5}, (p₂): {2,4}, (p₄): {1}, (p₅): {0}</td>
<td>(p₄)</td>
<td>(f)</td>
<td>(s₀)</td>
<td>(p₅)</td>
<td>(s₀)</td>
<td>(∅)</td>
<td>none</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(i)</td>
<td>(∅)</td>
<td>(∅)</td>
<td>(∅)</td>
<td>(∅)</td>
<td>none</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>(e)</td>
<td>(∅)</td>
<td>(∅)</td>
<td>(∅)</td>
<td>(∅)</td>
<td>none</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>(f)</td>
<td>(∅)</td>
<td>(∅)</td>
<td>(∅)</td>
<td>(∅)</td>
<td>none</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>(i)</td>
<td>(∅)</td>
<td>(∅)</td>
<td>(∅)</td>
<td>(∅)</td>
<td>none</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Reconstructing the DFA

- Each set in Partition forms a state
- For each line in the table where \(q₁ \neq ∅\), add an edge from \(q₁\) to \(s\) labelled \(c\)
### Reconstructing the DFA

- Each set in Partition forms a state
- For each line in the table where \( q_1 \neq \emptyset \), add an edge from \( q_1 \) to \( s \) labelled \( c \)
**DFA Minimization Algorithm (Worklist version)**

Worklist $\leftarrow \{ \mathcal{D}_A, \{ D - \mathcal{D}_A \} \} \\
Partition \leftarrow \{ \mathcal{D}_A, \{ D - \mathcal{D}_A \} \} \\

While (Worklist $\neq \emptyset$) do

   select a set $S$ from Worklist and remove it

   for each $\alpha \in \Sigma$ do

      $\text{Image} \leftarrow \{ x \mid \delta(x, \alpha) \in S \} \\

      for each $q \in \text{Partition that has a state in Image}$ do

         $q_1 \leftarrow q \cap \text{Image} \\
         q_2 \leftarrow q - q_1 \\
      
      if $q_2 \neq \emptyset$ then

         remove $q$ from Partition

         $\text{Partition} \leftarrow \text{Partition} \cup q_1 \cup q_2 \\

      if $q \in \text{Worklist}$ then

         remove $q$ from Worklist

         $\text{Worklist} \leftarrow \text{Worklist} \cup q_1 \cup q_2 \\

      else if $|q_1| \leq |q_2|$ then

         Worklist $\leftarrow \text{Worklist} \cup q_1 \\
      \text{else Worklist} \leftarrow \text{Worklist} \cup q_2 \\

   if $s = q$ then

      break;  // cannot keep working on $s$

---

**Why does this algorithm halt?**

- Fixed-point algorithm
- NFA has finite number of states
- Start with 2 sets in Partition
- Splitting breaks 1 set into 2 smaller ones but never makes a set larger
  $\rightarrow$ Monotone behavior
- Simple, finite limit on $|\text{Partition}|$; it cannot be $>|\text{States}|$
- Finite # steps, monotone increasing construction $\Rightarrow$ algorithm halts
DFA Minimization Algorithm (Worklist version)

\[
\text{Worklist} \leftarrow \{D_A, \{D - D_A\}\}
\]
\[
\text{Partition} \leftarrow \{D_A, \{D - D_A\}\}
\]

\[
\text{While } (\text{Worklist} \neq \emptyset) \text{ do}
\]
\[
\text{select a set } S \text{ from Worklist and remove it}
\]
\[
\text{for each } \alpha \in \Sigma \text{ do}
\]
\[
\text{Image} \leftarrow \{x \mid \delta(x, \alpha) \in S\}
\]
\[
\text{for each } q \in \text{Partition that has a state in Image do}
\]
\[
q_1 \leftarrow q \cap \text{Image}
\]
\[
q_2 \leftarrow q - q_1
\]
\[
\text{if } q_2 \neq \emptyset \text{ then}
\]
\[
\text{remove } q \text{ from Partition}
\]
\[
\text{Partition} \leftarrow \text{Partition} \cup q_1 \cup q_2
\]
\[
\text{if } q \in \text{Worklist then}
\]
\[
\text{remove } q \text{ from Worklist}
\]
\[
\text{Worklist} \leftarrow \text{Worklist} \cup q_1 \cup q_2
\]
\[
\text{else if } |q_1| \leq |q_2|
\]
\[
\text{then Worklist} \leftarrow \text{Worklist} \cup q_1
\]
\[
\text{else Worklist} \leftarrow \text{Worklist} \cup q_2
\]
\[
\text{if } s = q \text{ then}
\]
\[
\text{break; } // \text{ cannot keep working on } s
\]

One last hack ...
To make an implementation faster, it should maintain an efficient way to determine, for a given state, which set currently contains that state.
DFA Minimization

What about $a \ ( b \mid c \ )^*$?

From the subset construction:

<table>
<thead>
<tr>
<th>States</th>
<th>$\varepsilon$-closure($\text{Move}(s,\ast)$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DFA</td>
<td>NFA</td>
</tr>
<tr>
<td>$s_0$</td>
<td>$q_0$</td>
</tr>
<tr>
<td>$s_1$</td>
<td>$q_1, q_2, q_3$</td>
</tr>
<tr>
<td></td>
<td>$q_4, q_6, q_9$</td>
</tr>
<tr>
<td>$s_2$</td>
<td>$q_5, q_8, q_9$</td>
</tr>
<tr>
<td></td>
<td>$q_3, q_4, q_6$</td>
</tr>
<tr>
<td>$s_3$</td>
<td>$q_7, q_8, q_9$</td>
</tr>
<tr>
<td></td>
<td>$q_3, q_4, q_6$</td>
</tr>
</tbody>
</table>

From last lecture ...
DFA Minimization

Applying Hopcroft’s DFA minimization algorithm

| Current Partition | Split on |  
|------------------|---------|---|
| \(P_0\)          | a       | b  | c  |
| \(\{s_1, s_2, s_3\}\) \(\{s_0\}\) | none    | none | none |

It splits no states after the initial partition

\(\Rightarrow\) The minimal DFA has two states

\(\Rightarrow\) One for \(\{s_0\}\)

\(\Rightarrow\) One for \(\{s_1, s_2, s_3\}\)

Earlier, I suggested that a human would design a simpler automaton than Thompson’s construction & the subset construction did.

Minimizing that DFA produces exactly the DFA that I claimed a human would design!
The Plan for Scanner Construction

**RE → NFA** *(Thompson’s construction)*
- Build an NFA for each term in the RE
- Combine them in patterns that model the operators

**NFA → DFA** *(Subset construction)*
- Build a DFA that simulates the NFA

**DFA → Minimal DFA**
- Hopcroft’s algorithm
- Brzozowski’s algorithm

**Minimal DFA → Scanner**
- See § 2.5 in EaC2e

**DFA → RE**
- All pairs, all paths problem
- Union together paths from \( s_0 \) to a final state

---

**The Cycle of Constructions**

1. **RE**
2. **NFA**
3. **DFA**
   - **minimal DFA**
4. **Constructions**
5. **Scanner**
Brzozowski’s Algorithm for DFA Minimization

The Intuition

- The subset construction merges prefixes in the NFA

Arbitrary NFA for abc | bc | ad

Thompson’s construction would have built something with lots more $\varepsilon$-transitions and only one final state

Subset construction eliminates $\varepsilon$-transitions and merges the paths for $a$. It leaves duplicate tails, such as $bc$, intact.
Aside on **NFA** for **abc | bc | ad**

Thompson’s construction makes a slightly more complex NFA

- More states, more epsilon-transitions
- A single final state
Brzozowski’s Algorithm

Idea: Use The Subset Construction Twice

• For an NFA \( N \)
  – Let \( reverse(N) \) be the NFA constructed by making initial state final, adding a new start state with an \( \varepsilon \)-transition to each previously final state, and reversing the other edges
  – Let \( subset(N) \) be the DFA produced by the subset construction on \( N \)
  – Let \( reachable(N) \) be \( N \) after removing any states that are not reachable from the initial state

• Then,

\[
reachable\left(subset\left(reverse\left(reachable\left(subset\left(reverse(N)\right)\right)\right)\right)
\]

is a minimal DFA that implements \( N \) [Brzozowski, 1962]

Not everyone finds this result to be intuitive.
Neither algorithm dominates the other.
Brzozowski’s Algorithm

Step 1
• The subset construction on $\text{reverse}(\text{NFA})$ merges suffixes in original NFA

subset($\text{reverse}(\text{NFA})$)
Brzozowski’s Algorithm

Step 2

• Reverse it again & use subset to merge prefixes ...

Reverse it, again

And subset it, again

The Cycle of Constructions

Minimal DFA
Using **DFA** Minimization to Build a Scanner

**DFA minimization combines all the accepting (or final) states**

- Makes token identification much more difficult
- Cannot minimize the **DFA** for the entire set of words

---

**DFA from Lab 1 Lecture on Scanning**
Using **DFA** Minimization to Build a Scanner

**Cannot build and minimize one DFA for the entire set of words**

- Need to preserve final states
- Cannot create one giant **RE** and apply the cycle of constructions

**The solutions are fairly obvious**

*For Hopcroft’s algorithm:*

- Construct an initial partition that has a separate set for the final states of each syntactic category
- The algorithm will preserve that distinction

*For Brzozowski’s algorithm:*

- Create one or more **RE** for each syntactic category
- Build a minimal **DFA** for each of those **REs**
- Add a new start state that combines all the minimal **DFAs** & subset it
Using DFA Minimization to Build a Scanner

Cannot minimize the DFA for the entire set of words
• Need to preserve final states
• Cannot create one giant RE

The solutions are fairly obvious

For Hopcroft’s algorithm:
• Construct an initial partition that has a separate set for the final states of each syntactic category
• The algorithm will preserve that distinction

For Brzozowski’s algorithm:
• Create one or more RE for each syntactic category
• Build a minimal DFA for each of those REs
• Add a new start state that combines all the minimal DFAs & subset it

The Critical Point:
• The construction will build a minimal DFA
• The size of the minimal DFA relates to the language described by the RE, not the size of the RE
• The result is a DFA, so it has $O(1)$ cost per character
• The compiler writer can use the “most natural” or “intuitive” RE
Kleene’s Construction

for $i \leftarrow 0$ to $|D| - 1$; // label each immediate path
   for $j \leftarrow 0$ to $|D| - 1$;
      $R^0_{ij} \leftarrow \{ a \mid \delta(d_i, a) = d_j \}$;
      if $(i = j)$ then
         $R^0_{ii} = R^0_{ii} \cup \{\varepsilon\}$;

for $k \leftarrow 0$ to $|D| - 1$; // label nontrivial paths
   for $i \leftarrow 0$ to $|D| - 1$;
      for $j \leftarrow 0$ to $|D| - 1$;
         $R^k_{ij} \leftarrow R^{k-1}_{ik} (R^{k-1}_{kk})^* R^{k-1}_{kj} \cup R^{k-1}_{ij}$

$L \leftarrow \{\}$ // union labels of paths from
For each final state $s_i$ // $s_0$ to a final state $s_i$
   $L \leftarrow L \cup R^{|D|-1}_{0i}$

The Cycle of Constructions

Adaptation of all points, all paths, low cost algorithm

COMP 412, Fall 2019
Extra Slides

(more examples)
### A Detailed Example

<table>
<thead>
<tr>
<th>Step</th>
<th>Partition</th>
<th>W’list</th>
<th>s</th>
<th>c</th>
<th>Image</th>
<th>q</th>
<th>q₁</th>
<th>q₂</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$p_0: {3,5}, p_1: {0,1,2,4}$</td>
<td>$p_0, p_1$</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>1</td>
<td>$p_0: {3,5}, p_1: {0,1,2,4}$</td>
<td>$p_1$</td>
<td>$p_0$</td>
<td>$e$</td>
<td>$s_2, s_4$</td>
<td>$p_1$</td>
<td>$s_2, s_4$</td>
<td>$s_0, s_1$</td>
<td>split $p_1$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>$p_0: {3,5}, p_2: {2,4}, p_3: {0,1}$</td>
<td>$p_2, p_3$</td>
<td>$p_0$</td>
<td>$f$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>none</td>
</tr>
<tr>
<td>2</td>
<td>$p_0: {3,5}, p_2: {2,4}, p_3: {0,1}$</td>
<td>$p_3$</td>
<td>$p_2$</td>
<td>$e$</td>
<td>$s_1$</td>
<td>$p_3$</td>
<td>$s_1$</td>
<td>$s_0$</td>
<td>split $p_3$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>$p_0: {3,5}, p_2: {2,4}, p_4: {1}, p_5: {0}$</td>
<td>$p_4, p_5$</td>
<td>$p_2$</td>
<td>$f$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>$p_4, p_5$</td>
<td>$p_2$</td>
<td>$i$</td>
<td>$s_1$</td>
<td>$p_4$</td>
<td>$s_1$</td>
<td>$\emptyset$</td>
<td>none</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>$p_0: {3,5}, p_2: {2,4}, p_4: {1}, p_5: {0}$</td>
<td>$p_5$</td>
<td>$p_4$</td>
<td>$e$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>$p_5$</td>
<td>$p_4$</td>
<td>$f$</td>
<td>$s_0$</td>
<td>$p_5$</td>
<td>$s_0$</td>
<td>$\emptyset$</td>
<td>none</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$p_5$</td>
<td>$p_4$</td>
<td>$i$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>none</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>$p_0: {3,5}, p_2: {2,4}, p_4: {1}, p_5: {0}$</td>
<td>$\emptyset$</td>
<td>$p_5$</td>
<td>$e$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>$\emptyset$</td>
<td>$p_5$</td>
<td>$f$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>none</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$\emptyset$</td>
<td>$p_5$</td>
<td>$i$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>none</td>
<td></td>
</tr>
</tbody>
</table>
A DFA for \((a \mid b)^* abb\)

### NFA:

- \(s_0\) (Initial state)
- \(s_1\)
- \(s_2\)
- \(s_3\)
- \(s_4\) (Final state)

Transitions:
- \(s_0 \xrightarrow{\varepsilon} s_1\)
- \(s_1 \xrightarrow{a} s_2\)
- \(s_2 \xrightarrow{b} s_3\)
- \(s_3 \xrightarrow{b} s_4\)

### Subset Construction

<table>
<thead>
<tr>
<th>DFA state</th>
<th>NFA states</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>(d_0)</td>
<td>(s_0, s_1)</td>
<td>(s_0, s_1, s_2)</td>
<td>(d_0)</td>
</tr>
<tr>
<td>(d_1)</td>
<td>(s_0, s_1, s_2)</td>
<td>(d_1)</td>
<td>(s_0, s_1, s_3)</td>
</tr>
<tr>
<td>(d_2)</td>
<td>(s_0, s_1, s_3)</td>
<td>(d_1)</td>
<td>(s_0, s_1, s_4)</td>
</tr>
<tr>
<td>(d_3)</td>
<td>(s_0, s_1, s_2)</td>
<td>(d_1)</td>
<td>(d_0)</td>
</tr>
</tbody>
</table>

### DFA:

- \(d_0\) (Initial state)
- \(d_1\)
- \(d_2\)
- \(d_3\) (Final state)

Transitions:
- \(d_0 \xrightarrow{a} d_1\)
- \(d_1 \xrightarrow{a} d_2\)
- \(d_2 \xrightarrow{a} d_3\)
- \(d_3 \xrightarrow{b} d_3\)

- \(d_3 \xrightarrow{b} d_3\)

This DFA is already minimal. To create an example, we need to split one of the states so that the DFA is not minimal.
Another DFA for \((a | b)^* abb\)

- The **DFA** on previous slide is already minimal, by chance
- So, we can split \(d_0\) into \(s_0\) and \(s_2\) to create another version of the **DFA**
  - We need something to minimize
A Detailed Example

Splitting a Partition

- The algorithm starts out with \{ \{s_0, s_1, s_2, s_3\}, \{s_4\}\}
- How does \{s_4\} split \{s_0, s_1, s_2, s_3\}?
  - On \(a\), no edges run from \{s_0, s_1, s_2, s_3\} to \{s_4\}, so nothing splits
A Detailed Example

Splitting a Partition

• The algorithm starts out with \{ \{s_0, s_1, s_2, s_3\}, \{s_4\}\}

• How does \{s_4\} split \{s_0, s_1, s_2, s_3\}?

  – On \(b\), \{s_0, s_1, s_2, s_3\} has edges into both \{s_4\} and \{s_0, s_1, s_2, s_3\}, so \{s_4\} splits \{s_0, s_1, s_2, s_3\} into \{s_0, s_1, s_2\} and \{s_3\}

    \(\rightarrow\) \{s_0, s_1, s_2\} \Rightarrow \{s_0, s_1, s_2\} on \(b\)

    \(\rightarrow\) \{s_3\} \Rightarrow \{s_4\} on \(b\)
A Detailed Example

Splitting a Partition

• The algorithm starts out with \{ \{s_0, s_1, s_2, s_3\}, \{s_4\}\}.

• How does \{s_4\} split \{s_0, s_1, s_2, s_3\}?
  
  – On b, \{s_0, s_1, s_2, s_3\} has edges into both \{s_0, s_1, s_2, s_3\} and \{s_3\}.
  
  \[ \begin{align*}
  s_0 & \xrightarrow{a} s_1 \xrightarrow{b} s_3 \xrightarrow{a} s_4 \xrightarrow{b} s_2 \\
  s_2 & \xrightarrow{a} s_1 \xrightarrow{b} s_3 \xrightarrow{a} s_4 \xrightarrow{b} s_2 \\
  s_3 & \xrightarrow{a} s_4 \\
  \end{align*} \]

  \[\{s_0, s_1, s_2, s_3\} \rightarrow \{s_0, s_1, s_2\} \rightarrow \{s_0, s_1, s_2\} \text{ on } b
  \]

  \[\{s_3\} \rightarrow \{s_4\} \text{ on } b \]

  Note that when we split \{s_0, s_1, s_2, s_3\} around \{s_4\}, we left behind more work — the resulting set, \{s_0, s_1, s_2\}, could be split further.

  In the algorithm, \{s_3\} ends up on the worklist, where it will later split \{s_0, s_1, s_2\}.

Now, every state in \{s_3\} has the same transition on b

• Singleton set \Rightarrow same transition

• Neither \{s_3\} nor \{s_4\} can be split

• \{s_4\} causes no more splits

• \{s_3\} will split \{s_0, s_1, s_2\} into \{s_0, s_1\} and \{s_2\}
### Detailed Example

<table>
<thead>
<tr>
<th>Current Partition</th>
<th>Worklist</th>
<th>s</th>
<th>Split on a</th>
<th>Split on b</th>
</tr>
</thead>
<tbody>
<tr>
<td>{}</td>
<td>{{s_0}, {s_1}, {s_2}, {s_3}}</td>
<td>{s_4}</td>
<td>{{s_0}, {s_1}, {s_2}, {s_3}}</td>
<td></td>
</tr>
</tbody>
</table>

Example in this tabular format is for the worklist version of the algorithm.

![Diagram](image)
## Detailed Example

<table>
<thead>
<tr>
<th>Current Partition</th>
<th>Worklist</th>
<th>s</th>
<th>Split on a</th>
<th>Split on b</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>{s_4} {s_0,s_1,s_2,s_3}</td>
<td>{s_4} {s_0,s_1,s_2,s_3}</td>
<td>{s_4}</td>
<td>none</td>
</tr>
</tbody>
</table>

### Diagram

![Diagram of automaton with states: s_0, s_1, s_2, s_3, s_4, and transitions labeled with a, b.](image-url)
## Detailed Example

<table>
<thead>
<tr>
<th>Current Partition</th>
<th>Worklist</th>
<th>s</th>
<th>Split on a</th>
<th>Split on b</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>${s_4}$ ${s_0, s_1, s_2, s_3}$</td>
<td>$s_4$</td>
<td>${s_4}$</td>
<td>none ${s_3}$ ${s_0, s_1, s_2}$</td>
</tr>
</tbody>
</table>

![Transition Diagram](image)
### Detailed Example

<table>
<thead>
<tr>
<th>Current Partition</th>
<th>Worklist</th>
<th>s</th>
<th>Split on a</th>
<th>Split on b</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>{s_4} {s_0, s_1, s_2, s_3}</td>
<td>{s_4} {s_0, s_1, s_2, s_3}</td>
<td>{s_4}</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>{s_4} {s_3} {s_0, s_1, s_2}</td>
<td>{s_3} {s_0, s_1, s_2}</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Diagram:***

- **States:** \(s_0, s_1, s_2, s_3, s_4\)
- **Transitions:**
  - \(s_0 \xrightarrow{a} s_1\)
  - \(s_1 \xrightarrow{a} s_2\)
  - \(s_2 \xrightarrow{a} s_3\)
  - \(s_3 \xrightarrow{b} s_4\)
  - \(s_4 \xrightarrow{a} s_3\)

**Notes:**
- The worklist shows the current set of states to be processed.
- The partition shows the current division of states.
- The split conditions indicate whether to split on state or transition.

---

**Table Data:**

- **Row 0:**
  - **Current Partition:** \(\{s_4\} \{s_0, s_1, s_2, s_3\}\)
  - **Worklist:** \(\{s_4\} \{s_0, s_1, s_2, s_3\}\)
  - **Split on a:** \(\{s_4\}\)
  - **Split on b:** none

- **Row 1:**
  - **Current Partition:** \(\{s_4\} \{s_3\} \{s_0, s_1, s_2\}\)
  - **Worklist:** \(\{s_3\} \{s_0, s_1, s_2\}\)
  - **Split on a:** none
  - **Split on b:** \(\{s_3\} \{s_0, s_1, s_2\}\)
### Detailed Example

<table>
<thead>
<tr>
<th>Current Partition</th>
<th>Worklist</th>
<th>s</th>
<th>Split on a</th>
<th>Split on b</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>{s₄} {s₀,s₁,s₂,s₃}</td>
<td>{s₄} {s₀,s₁,s₂,s₃}</td>
<td>{s₄}</td>
<td>none</td>
</tr>
<tr>
<td>1</td>
<td>{s₄} {s₃} {s₀,s₁,s₂}</td>
<td>{s₃} {s₀,s₁,s₂}</td>
<td>{s₃}</td>
<td>none</td>
</tr>
</tbody>
</table>
### Detailed Example

<table>
<thead>
<tr>
<th></th>
<th>Current Partition</th>
<th>Worklist</th>
<th>s</th>
<th>Split on a</th>
<th>Split on b</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>{s_4} {s_0,s_1,s_2,s_3}</td>
<td>{s_4} {s_0,s_1,s_2,s_3}</td>
<td>{s_4}</td>
<td>none</td>
<td>{s_3} {s_0,s_1,s_2}</td>
</tr>
<tr>
<td>1</td>
<td>{s_4} {s_3} {s_0,s_1,s_2}</td>
<td>{s_3} {s_0,s_1,s_2}</td>
<td>{s_3}</td>
<td>none</td>
<td>{s_1} {s_0,s_2}</td>
</tr>
</tbody>
</table>

![Diagram](image)
## Detailed Example

<table>
<thead>
<tr>
<th></th>
<th>Current Partition</th>
<th>Worklist</th>
<th>s</th>
<th>Split on a</th>
<th>Split on b</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>{s_4} {s_0,s_1,s_2,s_3}</td>
<td>{s_4} {s_0,s_1,s_2,s_3}</td>
<td>{s_4}</td>
<td>none</td>
<td>{s_3} {s_0,s_1,s_2}</td>
</tr>
<tr>
<td>1</td>
<td>{s_4} {s_3} {s_0,s_1,s_2}</td>
<td>{s_3} {s_0,s_1,s_2}</td>
<td>{s_3}</td>
<td>none</td>
<td>{s_1} {s_0,s_2}</td>
</tr>
<tr>
<td>2</td>
<td>{s_4} {s_3} {s_1} {s_0,s_2}</td>
<td>{s_1} {s_0,s_2}</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Diagram:

![Transition Diagram](image-url)
### Detailed Example

<table>
<thead>
<tr>
<th>Current Partition</th>
<th>Worklist</th>
<th>s</th>
<th>Split on a</th>
<th>Split on b</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>{s_4} {s_0,s_1,s_2,s_3}</td>
<td>{s_4} {s_0,s_1,s_2,s_3}</td>
<td>{s_4}</td>
<td>none</td>
</tr>
<tr>
<td>1</td>
<td>{s_4} {s_3} {s_0,s_1,s_2}</td>
<td>{s_3} {s_0,s_1,s_2}</td>
<td>{s_3}</td>
<td>none</td>
</tr>
<tr>
<td>2</td>
<td>{s_4} {s_3} {s_1} {s_0,s_2}</td>
<td>{s_1} {s_0,s_2}</td>
<td>{s_1}</td>
<td>none</td>
</tr>
</tbody>
</table>

![State Transition Diagram]

---

COMP 412, Fall 2019
## Detailed Example

<table>
<thead>
<tr>
<th>Current Partition</th>
<th>Worklist</th>
<th>$s$</th>
<th>Split on $a$</th>
<th>Split on $b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>${s_4} {s_0,s_1,s_2,s_3}$</td>
<td>${s_4}$</td>
<td>none</td>
<td>${s_3} {s_0,s_1,s_2}$</td>
</tr>
<tr>
<td>1</td>
<td>${s_4} {s_3} {s_0,s_1,s_2}$</td>
<td>${s_3}$</td>
<td>none</td>
<td>${s_1} {s_0,s_2}$</td>
</tr>
<tr>
<td>2</td>
<td>${s_4} {s_3} {s_1} {s_0,s_2}$</td>
<td>${s_1}$</td>
<td>none</td>
<td>none</td>
</tr>
<tr>
<td>3</td>
<td>${s_4} {s_3} {s_1} {s_0,s_2}$</td>
<td>${s_0,s_2}$</td>
<td>none</td>
<td>none</td>
</tr>
</tbody>
</table>

Empty worklist $\Rightarrow$ done!

---

COMP 412, Fall 2019
Detailed Example

<table>
<thead>
<tr>
<th>Current Partition</th>
<th>Worklist</th>
<th>s</th>
<th>Split on a</th>
<th>Split on b</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>{s₄} {s₀,s₁,s₂,s₃}</td>
<td>{s₄} {s₀,s₁,s₂,s₃}</td>
<td>{s₄}</td>
<td>none</td>
</tr>
<tr>
<td>1</td>
<td>{s₄} {s₃} {s₀,s₁,s₂}</td>
<td>{s₃} {s₀,s₁,s₂}</td>
<td>{s₃}</td>
<td>none</td>
</tr>
<tr>
<td>2</td>
<td>{s₄} {s₃} {s₁} {s₀,s₂}</td>
<td>{s₁} {s₀,s₂}</td>
<td>{s₁}</td>
<td>none</td>
</tr>
<tr>
<td>3</td>
<td>{s₄} {s₃} {s₁} {s₀,s₂}</td>
<td>{s₀,s₂}</td>
<td>{s₀,s₂}</td>
<td>none</td>
</tr>
</tbody>
</table>

20% reduction in number of states
Abbreviated Register Specification

**Start with a regular expression**

\[ r0 \mid r1 \mid r2 \mid r3 \mid r4 \mid r5 \mid r6 \mid r7 \mid r8 \mid r9 \]

Register names from zero to nine

*The Cycle of Constructions*
Abbreviated Register Specification

Thompson’s construction produces

To make the example fit, we have eliminated some of the ε-transitions, e.g., between r and 0
Applying the subset construction yields

This is a **DFA**, but it has a lot of states ...

### The Cycle of Constructions

- RE → NFA → DFA → minimal DFA
Abbreviated Register Specification

Hopcroft’s algorithm

Initial sets

$F$ does not split.
Since no transitions leave it, there are no states to split it.

Technically, this edge shows up as 10 transitions, which are combined by construction of the character classifier ...

The Cycle of Constructions
Abbreviated Register Specification

Hopcroft’s algorithm

Initial sets

\[ S \rightarrow F \]

\{ S \rightarrow F \} does split

Any character in \( \Sigma \) will split it into \{ s_0 \}, \{ s_1 \}

The Cycle of Constructions

Technically, this edge shows up as 10 transitions, which are combined by construction of the character classifier ...
Hopcroft’s algorithm

Initial sets

\{ S \rightarrow F \} does split

Any character in \( \Sigma \) will split it into \{ s_0 \}, \{ s_1 \}

This partition is the final partition

Technically, this edge shows up as 10 transitions, which are combined by construction of the character classifier …
Abbreviated Register Specification

Hopcroft’s algorithm

Initial sets

The Critical Takeaway Points:
• The construction will build a minimal DFA
• The size of the DFA relates to the language described by the RE, not the size of the RE
• The result is a DFA, so it has $O(1)$ cost per character
• The compiler writer can use the most “natural” or “intuitive” RE

Becomes, through minimization

The Cycle of Constructions
Abbreviated Register Specification

Start with a regular expression

\[ r_0 \mid r_1 \mid r_2 \mid r_3 \mid r_4 \mid r_5 \mid r_6 \mid r_7 \mid r_8 \mid r_9 \]

Register names from zero to nine

The Cycle of Constructions
Thompson’s construction produces something along these lines

To make the example fit, we have eliminated some of the $\varepsilon$-transitions, e.g., between $r$ and 0

The Cycle of Constructions
Abbreviated Register Specification

Applying the subset construction yields

This is a **DFA**, but it has a lot of states ...
Abbreviated Register Specification

Applying Brzozowski’s algorithm, step 1

The Cycle of Constructions

COMP 412, Fall 2019
Brzozowski, step 2 reverses that DFA and subsets it again

A skilled human might build this DFA

The Critical Point:
• The construction will build a minimal DFA
• The size of the DFA relates to the language described by the RE, not the size of the RE
• The result is a DFA, so it has $O(1)$ cost per character
• The compiler writer can use the “most natural” or “intuitive” RE