Ignore § 2.4.4 in EaC2e. I will post a replacement section on the course web site.

Minor corrections applied after class

Copyright 2018, Keith D. Cooper & Linda Torczon, all rights reserved.
Students enrolled in Comp 412 at Rice University have explicit permission to make copies of these materials for their personal use.
Faculty from other educational institutions may use these materials for nonprofit educational purposes, provided this copyright notice is preserved.
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

---

COMP 412, Fall 2017
DFA Minimization

The Big Picture

• Discover sets of behaviorally equivalent states in the DFA
• Represent each such set with a single new state

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 DFA’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, optimistically

• Iteratively subdivide those sets, based on transition graph

• States that remain grouped together are equivalent

Initial partition: $P_0$ has two sets: $\{D_A\} \& \{D \setminus D_A\}$

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

• Assume $s_i \& s_j \in p_s$, and $\delta(s_i, a) = s_x$, & $\delta(s_j, a) = s_y$

• If $s_x \& s_y$ are not in the same set $p_t$, then $p_s$ must be split
  – 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)

Worklist ← \{D_A, \{D - D_A\}\}
Partition ← \{D_A, \{D - D_A\}\}

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

Image 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 \(S\)
Key Idea: Splitting Q Around Transitions on $\alpha$

Partitioning Q around $S$

As the algorithm considers $s$ and $\alpha$, it will split $q$.

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 $\alpha$. 
Key Idea: Splitting q around s and $\alpha$

Find maximal subset of q ($p_1$) that has an $\alpha$-transition into s

Think of $p_1$ as the image of s into q under the inverse of the transition function:

$$p_1 \leftarrow \delta^{-1}(s, \alpha) \cap q$$

$p_2$ 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 ← \{ D_A, \{ D - D_A \} \}
Partition ← \{ D_A, \{ D - D_A \} \}

While (Worklist ≠ \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 ← Partition \cup q_1 \cup q_2
            
            if \( q \in \text{Worklist} \) then
                remove \( q \) from Worklist
                Worklist ← Worklist \cup q_1 \cup q_2
            else if \( |q_1| \leq |q_2| \)
                then Worklist ← Worklist \cup q_1
                else Worklist ← Worklist \cup q_2
            
            if \( s = q \) then
                break; // cannot keep working on \( s \)

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

\( p_1 \) is the subset of \( q \) that transitions to \( S \) on \( \alpha \)
\( p_2 \) 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)

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
    $Image \leftarrow \{ x \mid \delta(x, \alpha) \in S \}$
    for each $q \in$ Partition that has a state in Image do
      $q_1 \leftarrow q \cap Image$
      $q_2 \leftarrow q - q_1$
      if $q_2 \neq \emptyset$ then
        remove $q$ from Partition
        $Partition \leftarrow Partition \cup q_1 \cup q_2$
      if $q \in$ 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 $\leftarrow Worklist \cup q_2$
      if $s = q$ then
        break; // cannot keep working on $s$

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>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>
</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>∅</td>
<td>∅</td>
<td>∅</td>
<td>∅</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>∅</td>
<td>∅</td>
<td>∅</td>
<td>∅</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_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>
</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>

---

![Diagram](image)
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>$p_0: {3,5}, p_2: {2,4}$</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>
<tr>
<td></td>
<td>$p_0: {3,5}, p_2: {2,4}, p_4: {1}$</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>$p_0: {3,5}, p_2: {2,4}, p_4: {1}, p_5: {0}$</td>
<td>$p_4, p_5$</td>
<td>$p_2$</td>
<td>i</td>
<td>$s_1$</td>
<td>$p_4$</td>
<td>$s_1$</td>
<td>$\emptyset$</td>
<td>none</td>
</tr>
<tr>
<td>3</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>(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>
<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_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>(i)</td>
<td>(s_1)</td>
<td>(p_4)</td>
<td>(s_1)</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>
<tr>
<td></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>(f)</td>
<td>(s_0)</td>
<td>(p_5)</td>
<td>(s_0)</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>$p₀: {3,5}, p₁: {0,1,2,4}$</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>$p₃$</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>$\emptyset$</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>
<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>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>$p₅$</td>
<td>$p₄$</td>
<td>$f$</td>
<td>$s₀$</td>
<td>$p₅$</td>
<td>$s₀$</td>
<td>$\emptyset$</td>
<td>none</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$p₅$</td>
<td>$p₄$</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 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>
<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>( p₀: {3,5}, p₂: {2,4}, p₄: {1}, p₅: {0} )</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>
</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>( \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}, p₄: {1}, p₅: {0} )</td>
<td>( p₅ )</td>
<td>( p₄ )</td>
<td>f</td>
<td>( s₀ )</td>
<td>( p₅ )</td>
<td>( s₀ )</td>
<td>( \emptyset )</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>i</td>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
<td>none</td>
</tr>
<tr>
<td>4</td>
<td>( p₀: {3,5}, p₂: {2,4}, p₄: {1}, p₅: {0} )</td>
<td>( \emptyset )</td>
<td>( p₅ )</td>
<td>e</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}, p₄: {1}, p₅: {0} )</td>
<td>( \emptyset )</td>
<td>( p₅ )</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}, p₄: {1}, p₅: {0} )</td>
<td>( \emptyset )</td>
<td>( p₅ )</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

Reconstructing the DFA

- Each set in Partition forms a state
- For each line in the table where both \( q_1 \) and \( s \) have values, add an edge from \( q_1 \) to \( s \) labelled \( c \)

<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>
<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>

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

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 $c$

<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>
<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>
<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>$p_0: {3,5}, p_2: {2,4}, p_4: {1}, p_5: {0}$</td>
<td>$p_4, p_5$</td>
<td>$p_2$</td>
<td>$i$</td>
<td>$s_1$</td>
<td>$p_4$</td>
<td>$s_1$</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>
<tr>
<td></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>$f$</td>
<td>$s_0$</td>
<td>$p_5$</td>
<td>$s_0$</td>
<td>$\emptyset$</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>$i$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>none</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$e$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>none</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$f$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>none</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$i$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</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 $q_1 \neq \emptyset$, add an edge from $q_1$ to $s$ labelled $c$.
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}
\begin{align*}
& \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}\\
& \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 \text{else if } |q_1| \leq |q_2|\\
& \quad \quad \quad \text{then } \text{Worklist} \leftarrow \text{Worklist} \cup q_1\\
& \quad \quad \quad \text{else } \text{Worklist} \leftarrow \text{Worklist} \cup q_2\\
& \quad \quad \text{if } s = q \text{ then}\\
& \quad \quad \quad \text{break; } // \text{ cannot keep working on } s
\end{align*}

Why does this algorithm halt?
\begin{itemize}
\item Fixed-point algorithm
\item DFA has finite number of states
\item Start with 2 sets in Partition
\item Splitting breaks 1 set into 2 smaller ones but never makes a set larger \text{ Monotone behavior}
\item Simple, finite limit on \( |\text{Partition} |; \text{ it cannot be } > |\text{States} | \)
\item Finite \# steps, monotone increasing construction \Rightarrow \text{ algorithm halts}
\end{itemize}
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
    $Image \leftarrow \{ x \mid \delta(x, \alpha) \in S \}$
    for each $q \in$ Partition that has a state in $Image$ do
      $q_1 \leftarrow q \cap Image$
      $q_2 \leftarrow q - q_1$
      if $q_2 \neq \emptyset$ then
        remove $q$ from Partition
        Partition $\leftarrow$ Partition $\cup$ $q_1 \cup q_2$
      if $q \in$ 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 $\cup$ $q_2$
    if $s = q$ then
      break; // 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 contain that state.
DFA Minimization

What about \( a ( b | c )^* \)?

From the subset construction:

<table>
<thead>
<tr>
<th>States</th>
<th>DFA</th>
<th>NFA</th>
<th>( \varepsilon )-closure(Move(s,*))</th>
</tr>
</thead>
<tbody>
<tr>
<td>( s_0 )</td>
<td>( q_0 )</td>
<td></td>
<td>( s_1 )</td>
</tr>
<tr>
<td>( s_1 )</td>
<td>( q_1, q_2, q_3, q_4, q_6, q_9 )</td>
<td></td>
<td>none</td>
</tr>
<tr>
<td>( s_2 )</td>
<td>( q_5, q_8, q_9 )</td>
<td></td>
<td>none</td>
</tr>
<tr>
<td>( s_3 )</td>
<td>( q_7, q_8, q_9 )</td>
<td></td>
<td>none</td>
</tr>
</tbody>
</table>

From last lecture ...
### DFA Minimization

#### Applying Hopcroft’s DFA minimization algorithm

<table>
<thead>
<tr>
<th>Current Partition</th>
<th>Split on</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a</td>
</tr>
<tr>
<td>$P_0$</td>
<td>none</td>
</tr>
</tbody>
</table>

- $\{s_1, s_2, s_3\} \cup \{s_0\}$

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!
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

COMP 412, Fall 2017
Abbreviated Register Specification

Thompson’s construction produces

![Diagram of Thompson's construction]

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

The Cycle of Constructions

COMP 412, Fall 2017
Applying the subset construction yields

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

Abbreviated Register Specification

The Cycle of Constructions
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...
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 ...
Abbreviated Register Specification

Hopcroft’s algorithm

Initial sets

\[ \{ S \rightarrow F \} \text{ does split} \]

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

This partition is the final partition

The Cycle of Constructions

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

Becomes, through minimization

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

The Cycle of Constructions
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
Brzozowski’s Algorithm for DFA Minimization

The Intuition

- The subset construction merges prefixes in the NFA

Thompson’s construction would leave $\varepsilon$-transitions between each single-character automaton

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

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

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

![Graph of Brzozowski’s Algorithm]

Reversed NFA

subset(reverse(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

COMP 412, Fall 2017
Abbreviated Register Specification

Start with a regular expression
r0 | r1 | r2 | r3 | r4 | r5 | r6 | r7 | r8 | r9

Register names from zero to nine

The Cycle of Constructions

COMP 412, Fall 2017
Abbreviated Register Specification

Thompson’s construction produces something along these lines

\[ r^0 r^1 r^2 \ldots \]

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

The Cycle of Constructions
Applying the subset construction yields

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

**The Cycle of Constructions**
Abbreviated Register Specification

Applying Brzozowski’s algorithm, step 1

Reversed NFA

After Subset Construction

The Cycle of Constructions
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
Kleene’s Construction

```plaintext
for i ← 0 to |D| - 1; // label each immediate path
    for j ← 0 to |D| - 1;
        R^0_{ij} ← \{ a | \delta(d_i, a) = d_j \};
        if (i = j) then
            R^0_{ii} = R^0_{ii} | \{\varepsilon\};
    
for k ← 0 to |D| - 1; // label nontrivial paths
    for i ← 0 to |D| - 1;
        for j ← 0 to |D| - 1;
            R^k_{ij} ← R^{k-1}_{ik} (R^{k-1}_{kk})^* R^{k-1}_{kj} | R^{k-1}_{ij}

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

The Wikipedia page on “Kleene’s algorithm” is pretty good. It also contains a link to Kleene’s 1956 paper. This form of the algorithm is usually attributed to McNaughton and Yamada in 1960.

Adaptation of all points, all paths, low cost algorithm

COMP 412, Fall 2017