Automata

Compiler Design

CSE 504
NFA to DFA via Subset Construction

Diagram:

- Start state: 0
- States: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13
- Transitions:
  - From 0 to 1, 2, 3, 4, 5, 6, 7
  - From 1 to 2, 3, 4, 5, 6, 7
  - From 2 to 3
  - From 3 to 2, 3, 5
  - From 4 to 6, 7
  - From 5 to 6
  - From 6 to 3, 7
  - From 7 to 8, 9
  - From 8 to 9, 11
  - From 9 to 10, 11
  - From 10 to 12
  - From 11 to 12
  - From 12 to 13
  - From 13 to 10
  - From 8 to 7 (ε-transitions)

Final states: 9, 11, 13

NFA to DFA via Subset Construction

\[ \epsilon \text{-closure} \]
NFA to DFA via Subset Construction

\[ \epsilon \text{-closure} \]
NFA to DFA via Subset Construction
NFA to DFA via Subset Construction

\[ \epsilon \text{-closure} \]
NFA to DFA via Subset Construction

$\epsilon$-closure
NFA to DFA via Subset Construction

ω

goto

ω
NFA to DFA via Subset Construction

\[\begin{align*}
0 & \rightarrow 1 \\
1 & \rightarrow 2, 3 \\
2 & \rightarrow 4 \\
3 & \rightarrow 5 \\
4 & \rightarrow 6, 9 \\
5 & \rightarrow 6 \\
6 & \rightarrow 7, 8 \\
7 & \rightarrow 8 \\
8 & \rightarrow 9, 11, 13 \\
9 & \rightarrow 10, 11 \\
10 & \rightarrow 11, 12, 13 \\
11 & \rightarrow 12, 13 \\
12 & \rightarrow 13 \\
13 & \rightarrow 9, 11
\end{align*}\]
NFA to DFA via Subset Construction

\[ \text{\(\varepsilon\)-closure} \]
NFA to DFA via Subset Construction

goto
NFA to DFA via Subset Construction

\[ \varepsilon \text{-closure} \]
NFA to DFA via Subset Construction

- ε-closure
NFA to DFA via Subset Construction

goto
NFA to DFA via Subset Construction

\[ \epsilon \text{-closure} \]
NFA to DFA via Subset Construction

\(\epsilon\)-closure

\[0, 1, 2, 3, 7\]
\[4, 8, 1, 2, 3, 6, 7, 9, 10\]
\[4, 8, 11, 1, 2, 3, 6, 7, 9, 10, 13\]
\[5, 1, 2, 3, 6, 7\]
\[5, 12, 1, 2, 3, 6, 7, 13\]
NFA to DFA via Subset Construction

goto
NFA to DFA via Subset Construction

Final states
Finite Automata and Languages

Consider a finite automaton $A$.

- For state $s$ and word $w$, let $s.w$ be the state reached from $s$ by spelling $w$.
  
  Note that a word $w$ is a sequence of (zero or more) alphabet symbols.
Consider a finite automaton $A$.

- For state $s$ and word $w$, let $s.w$ be the state reached from $s$ by spelling $w$.
  Note that a word $w$ is a sequence of (zero or more) alphabet symbols.
- We can associate two languages with each state in $A$: 

Consider a finite automaton $A$.

- For state $s$ and word $w$, let $s.w$ be the state reached from $s$ by spelling $w$.
  Note that a word $w$ is a sequence of (zero or more) alphabet symbols.

- We can associate two languages with each state in $A$:
  - **Suffix:** For state $s$, define $L_{s \rightarrow} = \{ w | s.w \text{ is a final state} \}$
Finite Automata and Languages

Consider a finite automaton $A$.

- For state $s$ and word $w$, let $s.w$ be the state reached from $s$ by spelling $w$.
  Note that a word $w$ is a sequence of (zero or more) alphabet symbols.
- We can associate two languages with each state in $A$:
  - **Suffix**: For state $s$, define $L_s \rightarrow = \{ w | s.w \text{ is a final state} \}$
  - **Prefix**: Let $s$ be the start state of $A$. For a state $t$, define $L_{\rightarrow s} = \{ w | s.w = t \}$. 


Finite Automata and Languages

Consider a finite automaton $A$.

- For state $s$ and word $w$, let $s.w$ be the state reached from $s$ by spelling $w$. Note that a word $w$ is a sequence of (zero or more) alphabet symbols.
- We can associate two languages with each state in $A$:
  - **Suffix**: For state $s$, define $L_s \rightarrow = \{ w | s.w \text{ is a final state} \}$
  - **Prefix**: Let $s$ be the start state of $A$. For a state $t$, define $L_{\rightarrow s} = \{ w | s.w = t \}$.
- The language of $A$, denoted by $L_A$ can be seen as:
Consider a finite automaton $A$.

- For state $s$ and word $w$, let $s.w$ be the state reached from $s$ by spelling $w$.
  Note that a word $w$ is a sequence of (zero or more) alphabet symbols.

- We can associate two languages with each state in $A$:
  - **Suffix**: For state $s$, define $L_{s \rightarrow} = \{ w \mid s.w \text{ is a final state} \}$
  - **Prefix**: Let $s$ be the start state of $A$. For a state $t$, define $L_{\rightarrow s} = \{ w \mid s.w = t \}$.

- The language of $A$, denoted by $\mathcal{L}_A$ can be seen as:
  - $L_{s \rightarrow}$ where $s$ is the start state;
Consider a finite automaton $A$.

- For state $s$ and word $w$, let $s.w$ be the state reached from $s$ by spelling $w$.
  - Note that a word $w$ is a sequence of (zero or more) alphabet symbols.
- We can associate two languages with each state in $A$:
  - **Suffix**: For state $s$, define $L_{s \rightarrow} = \{ w | s.w \text{ is a final state} \}$
  - **Prefix**: Let $s$ be the start state of $A$. For a state $t$, define $L_{\rightarrow s} = \{ w | s.w = t \}$.
- The language of $A$, denoted by $L_A$ can be seen as:
  - $L_{s \rightarrow}$ where $s$ is the start state;
  - $\bigcup_f$ is a final state $L_{\rightarrow f}$
Examples of Suffix and Prefix Languages (1)

- $L_{s_0\rightarrow} = (ab)^+$
- $L_{s_1\rightarrow} = b(ab)^*$
- $L_{s_2\rightarrow} = (ab)^*$
- $L_{\rightarrow s_1} = a(ba)^*$
- $L_{\rightarrow s_2} = (ab)^+$

Diagram:

- $s_0 \rightarrow a \rightarrow s_1 \rightarrow a \rightarrow b \rightarrow s_2$
Examples of Suffix and Prefix Languages (2)

\[ L_{s_1} = (a|b)^* \]
\[ L_{s_2} = (a|b)^* \]
\[ L_{\rightarrow s_1} = a(a|b)^* \]
\[ L_{\rightarrow s_2} = (a+)b(a|b)^* \]
An automaton is *minimal* if the suffix languages of all states are pairwise distinct.
An automaton is *minimal* if the suffix languages of all states are pairwise distinct.

The following automaton is not minimal:

\[
\begin{array}{c}
s_0 \xrightarrow{a} s_1 \xrightarrow{b} s_2 \xrightarrow{a} s_1 \\
L_{s_1} = L_{s_2} = (a|b)^* 
\end{array}
\]
An automaton is \textit{minimal} if the suffix languages of all states are pairwise distinct.

The following automaton is \textbf{not minimal}:

![Diagram of an automaton with states s0, s1, and s2, labeled with transitions a and b.]

$L_{s_1 \rightarrow} = L_{s_2 \rightarrow} = (a|b)^*$

The following automaton is \textbf{minimal}:

![Diagram of a minimal automaton with states s0, s1, and s2, labeled with transitions a and b.]

The suffix languages of all states are distinct. Recall:

$L_{s_0 \rightarrow} = (ab)^+; \quad L_{s_1 \rightarrow} = b(ab)^*; \quad L_{s_2 \rightarrow} = (ab)^*$
Equivalence and Minimality (2)

Minimization approach: find the *coarsest* partition such that

- the suffix languages of states within a partition are identical; and
Equivalence and Minimality (2)

Minimization approach: find the *coarsest* partition such that

- the suffix languages of states within a partition are identical; and
- states in distinct partitions have distinct suffix languages.
Equivalence and Minimality (2)

Minimization approach: find the *coarsest* partition such that
- the suffix languages of states within a partition are identical; and
- states in distinct partitions have distinct suffix languages.
Equivalence and Minimality (2)

Minimization approach: find the *coarsest* partition such that
- the suffix languages of states within a partition are identical; and
- states in distinct partitions have distinct suffix languages.

Original automaton:

\[
\begin{align*}
L_{s_0} & = a(a|b)^*; \\
L_{s_1} & = L_{s_2} = (a|b)^*; \\
L_{s_0} & = a(a|b)^*
\end{align*}
\]
**Equivalence and Minimality (2)**

Minimization approach: find the *coarsest* partition such that
- the suffix languages of states within a partition are identical; and
- states in distinct partitions have distinct suffix languages.

Original automaton:

![Original automaton diagram](image)

Minimized automaton:

![Minimized automaton diagram](image)
Partition Refinement

- Suffix languages of all final states are distinct from those of non-final states.
  - **Note:** $\epsilon$, the empty string, is in the suffix language of a final state, and not in that of a non-final state.
- We start by partitioning the states into final and non-final states.
- At each step, we refine a partition $P$ (if possible) if transitions on the same symbol from different states of $P$ lead to states in different partitions.
- The process stops when no partition can be refined further.
DFA Minimization via Partition Refinement

- Minimization via Partition Refinement

Diagram:

- States: d0, d1, d2, d3, d4
- Transitions:
  - d0 → a → d1 → a → d3
  - b → d2 → b
  - a → d1 → a
  - a → d3
  - b → d2 → b
  - b → d4 → b

Initial partition: {d0, d2}, {d1}, {d3}, {d4}

Final states: {d3, d4}

Non-final states: {d0, d1, d2}

(distinguished by ϵ)

- goto (d3, a) = d3 ∈ {d3, d4}
  - but goto (d4, a) = d1 \̸∈ {d3, d4}.

i.e., a ∈ Ld3 → but a \̸∈ Ld4 →.

New partitions:

- {d3}
- {d4}
- {d0, d2, d1}

- goto (d1, a) = d3,
  - but goto ({d0, d2}, a) = d1.

i.e., a ∈ Ld1 → but a \̸∈ Ld2 →.

New partitions:

- {d3}
- {d4}
- {d1}
- {d0, d2}
DFA Minimization via Partition Refinement

- Minimization via Partition Refinement
- Initial partition:
  - Final states: \( \{d3, d4\} \)
  - Non-final states: \( \{d0, d1, d2\} \)
  - (distinguished by \( \epsilon \))
DFA Minimization via Partition Refinement

- Minimization via Partition Refinement
- Initial partition:
  - Final states: \{d3, d4\}
  - Non-final states: \{d0, d1, d2\}
  (distinguished by \(\epsilon\))
- \(\text{goto}(d3, a) = d3 \in \{d3, d4\}\), but \(\text{goto}(d4, a) = d1 \notin \{d3, d4\}\).
  i.e., \(a \in L_{d3}\) but \(a \notin L_{d4}\).
DFA Minimization via Partition Refinement

- Minimization via Partition Refinement
- Initial partition:
  - Final states: \{d_3, d_4\}
  - Non-final states: \{d_0, d_1, d_2\}
  - (distinguished by \(\epsilon\))
- \(\text{goto}(d_3, a) = d_3 \in \{d_3, d_4\}\),
  - but \(\text{goto}(d_4, a) = d_1 \not\in \{d_3, d_4\}\).
  - i.e., \(a \in L_{d_3}\rightarrow\) but \(a \not\in L_{d_4}\rightarrow\).
- New partitions:
  - \{d_3\}; \{d_4\}; \{d_0, d_1, d_2\}
DFA Minimization via Partition Refinement

- Minimization via Partition Refinement
- Initial partition:
  - Final states: \{d3, d4\}
  - Non-final states: \{d0, d1, d2\}
  (distinguished by \(\epsilon\))
  - goto(d3, a) = d3 \(\in\) \{d3, d4\}, but goto(d4, a) = d1 \(\not\in\) \{d3, d4\}.
  i.e., a \(\in\) \(L_{d3\rightarrow}\) but a \(\not\in\) \(L_{d4\rightarrow}\).
- New partitions:
  - \{d3\}; \{d4\}; \{d0, d1, d2\}
  - goto(d1, a) = d3, but goto(\{d0, d2\}, a) = d1.
  i.e., a \(\in\) \(L_{d1\rightarrow}\) but a \(\not\in\) \(L_{d2\rightarrow}\).
DFA Minimization via Partition Refinement

- Minimization via Partition Refinement
- Initial partition:
  - Final states: \{d3, d4\}
  - Non-final states: \{d0, d1, d2\}
  - (distinguished by \(\epsilon\))

- \(\text{goto}(d3, a) = d3 \in \{d3, d4\}\), but \(\text{goto}(d4, a) = d1 \not\in \{d3, d4\}\).
  - i.e., \(a \in L_{d3\rightarrow}\) but \(a \not\in L_{d4\rightarrow}\).
- New partitions:
  - \{d3\}; \{d4\}; \{d0, d1, d2\}

- \(\text{goto}(d1, a) = d3\),
  - but \(\text{goto}(\{d0, d2\}, a) = d1\).
  - i.e., \(a \in L_{d1\rightarrow}\) but \(a \not\in L_{d2\rightarrow}\).
- New partitions:
  - \{d3\}; \{d4\}; \{d1\}; \{d0, d2\}
DFA Minimization via Partition Refinement

- Minimization via Partition Refinement
- Initial partition:
  - Final states: \( \{d3, d4\} \)
  - Non-final states: \( \{d0, d1, d2\} \)
    (distinguished by \( \epsilon \))
  - \( \text{goto}(d3, a) = d3 \in \{d3, d4\} \),
    but \( \text{goto}(d4, a) = d1 \notin \{d3, d4\} \).
    i.e., \( a \in L_{d3} \rightarrow \) but \( a \notin L_{d4} \rightarrow \).
- New partitions:
  - \( \{d3\}; \{d4\}; \{d0, d1, d2\} \)
  - \( \text{goto}(d1, a) = d3, \)
    but \( \text{goto}(\{d0, d2\}, a) = d1 \).
    i.e., \( a \in L_{d1} \rightarrow \) but \( a \notin L_{d2} \rightarrow \).
- New partitions:
  - \( \{d3\}; \{d4\}; \{d1\}; \{d0, d2\} \)
Moore’s Algorithm for DFA Minimization

Let $P$ be a partition of the set of states. States $s$ and $s'$ are equivalent in $P$ if $s$ and $s'$ are in the same set in $P$. 
Moore’s Algorithm for DFA Minimization

- Let $P$ be a partition of the set of states. States $s$ and $s'$ are equivalent in $P$ if $s$ and $s'$ are in the same set in $P$.
- Two states $s_1$ and $s_2$ are equivalent in $\alpha^{-1}.P$ if $s_1.\alpha$ and $s_2.\alpha$ are equivalent in $P$. 

![Diagram](image-url)
Moore’s Algorithm for DFA Minimization

- Let $P$ be a partition of the set of states.
  States $s$ and $s'$ are equivalent in $P$ if $s$ and $s'$ are in the same set in $P$.
- Two states $s_1$ and $s_2$ are equivalent in $\alpha^{-1}.P$ if $s_1.\alpha$ and $s_2.\alpha$ are equivalent in $P$.

\[ P_0: \{d3, d4\}; \{d0, d1, d2\} \]
Moore’s Algorithm for DFA Minimization

- Let $P$ be a partition of the set of states. States $s$ and $s'$ are equivalent in $P$ if $s$ and $s'$ are in the same set in $P$.
- Two states $s_1$ and $s_2$ are equivalent in $\alpha^{-1}.P$ if $s_1.\alpha$ and $s_2.\alpha$ are equivalent in $P$.

- $P_0$: $\{d3, d4\}; \{d0, d1, d2\}$
- Observe:
Moore’s Algorithm for DFA Minimization

Let $P$ be a partition of the set of states. States $s$ and $s'$ are equivalent in $P$ if $s$ and $s'$ are in the same set in $P$.

Two states $s_1$ and $s_2$ are equivalent in $\alpha^{-1}.P$ if $s_1.\alpha$ and $s_2.\alpha$ are equivalent in $P$.

- $P_0$: \{d3, d4\}; \{d0, d1, d2\}
- Observe:
  - $a^{-1}.P_0 = \{d1, d3\}; \{d0, d2, d4\}$
Moore’s Algorithm for DFA Minimization

- Let $P$ be a partition of the set of states. States $s$ and $s'$ are equivalent in $P$ if $s$ and $s'$ are in the same set in $P$.
- Two states $s_1$ and $s_2$ are equivalent in $\alpha^{-1}.P$ if $s_1.\alpha$ and $s_2.\alpha$ are equivalent in $P$.

- $P_0$: $\{d3, d4\};\{d0, d1, d2\}$
- Observe:
  - $a^{-1}.P_0 = \{d1, d3\};\{d0, d2, d4\}$
  - $b^{-1}.P_0 = \{d1, d3\};\{d0, d2, d4\}$
Moore’s Algorithm for DFA Minimization

- Let \( P \) be a partition of the set of states.
  States \( s \) and \( s' \) are equivalent in \( P \) if \( s \) and \( s' \) are in the same set in \( P \).
- Two states \( s_1 \) and \( s_2 \) are equivalent in \( \alpha^{-1}.P \) if \( s_1.\alpha \) and \( s_2.\alpha \) are equivalent in \( P \).

\[ P_0: \{d3, d4\}; \{d0, d1, d2\} \]

Observe:
- \( a^{-1}.P_0 = \{d1, d3\}; \{d0, d2, d4\} \)
- \( b^{-1}.P_0 = \{d1, d3\}; \{d0, d2, d4\} \)

\( P \land P' :: \) a partition such that two states are in same set iff they are in the same sets in \( P \) as well as \( P' \).
Moore’s Algorithm for DFA Minimization

- Let $P$ be a partition of the set of states. States $s$ and $s'$ are equivalent in $P$ if $s$ and $s'$ are in the same set in $P$.
- Two states $s_1$ and $s_2$ are equivalent in $\alpha^{-1}.P$ if $s_1.\alpha$ and $s_2.\alpha$ are equivalent in $P$.

- $P_0$: $\{d3, d4\}; \{d0, d1, d2\}$
- Observe:
  - $a^{-1}.P_0 = \{d1, d3\}; \{d0, d2, d4\}$
  - $b^{-1}.P_0 = \{d1, d3\}; \{d0, d2, d4\}$
- $P \land P'$ :: a partition such that two states are in same set iff they are in the same sets in $P$ as well as $P'$.
- $P_1 = P_0 \land a^{-1}.P_0 \land b^{-1}.P_0$
  - $= \{d0, d2\}; \{d1\}; \{d3\}; \{d4\}$
Moore’s Algorithm for DFA Minimization

- Let $P$ be a partition of the set of states. States $s$ and $s'$ are equivalent in $P$ if $s$ and $s'$ are in the same set in $P$.
- Two states $s_1$ and $s_2$ are equivalent in $\alpha^{-1}.P$ if $s_1.\alpha$ and $s_2.\alpha$ are equivalent in $P$.

\[
P_0: \{d3, d4\}; \{d0, d1, d2\}
\]

Observe:

- $a^{-1}.P_0 = \{d1, d3\}; \{d0, d2, d4\}$
- $b^{-1}.P_0 = \{d1, d3\}; \{d0, d2, d4\}$

$P \land P' ::$ a partition such that two states are in same set iff they are in the same sets in $P$ as well as $P'$.

$P_1 = P_0 \land a^{-1}.P_0 \land b^{-1}.P_0$

$= \{d0, d2\}; \{d1\}; \{d3\}; \{d4\}$

- In general $P_{i+1} = P_i \land_{\alpha \in \Sigma} \alpha^{-1}.P_i$
Moore’s Algorithm for DFA Minimization

- Let $P$ be a partition of the set of states. States $s$ and $s'$ are equivalent in $P$ if $s$ and $s'$ are in the same set in $P$.
- Two states $s_1$ and $s_2$ are equivalent in $\alpha^{-1}.P$ if $s_1.\alpha$ and $s_2.\alpha$ are equivalent in $P$.

\[ P_0: \{d3, d4\}; \{d0, d1, d2\} \]

- Observe:
  \[ a^{-1}.P_0 = \{d1, d3\}; \{d0, d2, d4\} \]
  \[ b^{-1}.P_0 = \{d1, d3\}; \{d0, d2, d4\} \]

- $P \land P'$ :: a partition such that two states are in same set iff they are in the same sets in $P$ as well as $P'$.

\[ P_1 = P_0 \land a^{-1}.P_0 \land b^{-1}.P_0 = \{d0, d2\}; \{d1\}; \{d3\}; \{d4\} \]

- In general $P_{i+1} = P_i \land_{\alpha \in \Sigma} \alpha^{-1}.P_i$
- Repeat until $P_{n+1} = P_n$
Another Example of Moore’s Algorithm

$P_0$: $s_0 \xrightarrow{a,b} s_1 \xrightarrow{a,b} s_2 \xrightarrow{a} s_3$
Another Example of Moore’s Algorithm

- $P_0$: 
  
  - $a^{-1}.P_0 = \{s0, s1\}; \{s2, s3\}$ and $b^{-1}.P_0 = \{s0, s1\}; \{s2\}; \{s3\}$
Another Example of Moore’s Algorithm

- $P_0$: $P_0 = \{s_0, s_1\}; \{s_2, s_3\}$ and $a^{-1}.P_0 = \{s_0, s_1\}; \{s_2, s_3\}$
- $P_1$: $P_1 = \{s_0, s_1\}; \{s_2\}; \{s_3\}$
Another Example of Moore’s Algorithm

- \( P_0: \)
  \[
  s_0 \xrightarrow{a,b} s_1 \xrightarrow{a,b} s_2 \xrightarrow{a} s_3
  \]
  \( a^{-1}.P_0 = \{s_0, s_1\}; \{s_2, s_3\} \) and \( b^{-1}.P_0 = \{s_0, s_1\}; \{s_2\}; \{s_3\} \)

- \( P_1: \)
  \[
  s_0 \xrightarrow{a,b} s_1 \xrightarrow{a,b} s_2 \xrightarrow{a} s_3
  \]
  \( a^{-1}.P_1 = \{s_0\}; \{s_1\}; \{s_2, s_3\} \) and \( b^{-1}.P_0 = \{s_0\}; \{s_1\}; \{s_2\}; \{s_3\} \)
Another Example of Moore’s Algorithm

- $P_0$: $s_0 \xrightarrow{ab} s_1 \xrightarrow{ab} s_2 \xrightarrow{a} s_3$
  
  $a^{-1}.P_0 = \{s_0, s_1\}; \{s_2, s_3\}$ and $b^{-1}.P_0 = \{s_0, s_1\}; \{s_2\}; \{s_3\}$

- $P_1$: $s_0 \xrightarrow{ab} s_1 \xrightarrow{ab} s_2 \xrightarrow{a} s_3$
  
  $a^{-1}.P_1 = \{s_0\}; \{s_1\}; \{s_2, s_3\}$ and $b^{-1}.P_0 = \{s_0\}; \{s_1\}; \{s_2\}; \{s_3\}$

- $P_2$: $s_0 \xrightarrow{ab} s_1 \xrightarrow{ab} s_2 \xrightarrow{a} s_3$
  
  $a^{-1}.P_1 = \{s_0\}; \{s_1\}; \{s_2, s_3\}$ and $b^{-1}.P_0 = \{s_0\}; \{s_1\}; \{s_2\}; \{s_3\}$
Yet Another Example of Moore’s Algorithm

\[ P_0 = \{1, 2, 3, 4, 7\}; \{5, 6\} \]
\[ a^{-1}.P_0 = \{1, 2, 3, 4, 6\}; \{5, 7\} \]
\[ b^{-1}.P_0 = \{1, 2, 4, 5, 6, 7\}; \{3\} \]

\[ P_1 = \{1, 2, 4\}; \{3\}; \{5\}, \{6\}, \{7\} \]
\[ a^{-1}.P_1 = \{1\}; \{2, 3, 4\}; \{5, 7\}; \{6\} \]
\[ b^{-1}.P_1 = \{1, 7\}; \{2, 4, 5, 6\}; \{3\} \]

\[ P_2 = \{1\}; \{2, 4\}; \{3\}; \{5\}, \{6\}, \{7\} \]
\[ a^{-1}.P_2 = \{1\}; \{2, 3, 4\}; \{5, 7\}; \{6\} \]
\[ b^{-1}.P_2 = \{1, 7\}; \{2, 4, 5, 6\}; \{3\} \]

\[ P_3 = P_2 \]
Brzozowski’s Algorithm for DFA Minimization

- Let $M = \text{subset}(\text{reverse}(\text{subset}(\text{reverse}(A))))$
- $M$ is a minimal automaton equivalent to $A$. 
Brzozowski’s Algorithm: Why it works

- Let $A$ be an NFA/DFA;
  - Let $B$ be an automaton generated by the subset construction algorithm (NFA to DFA) of $\text{reverse}(A)$.
Brzozowski’s Algorithm: Why it works

- Let $A$ be an NFA/DFA;
  Let $B$ be an automaton generated by the subset construction algorithm (NFA to DFA) of $\text{reverse}(A)$.
- Let $s_1$ and $s_2$ be two distinct states in $B$.
  Then, $L \rightarrow_{s_1} \cap L \rightarrow_{s_2} = \emptyset$.
Brzozowski’s Algorithm: Why it works

- Let $A$ be an NFA/DFA;
  Let $B$ be an automaton generated by the subset construction algorithm (NFA to DFA) of $\text{reverse}(A)$.
- Let $s_1$ and $s_2$ be two distinct states in $B$.
  Then, $L_{s_1} \cap L_{s_2} = \emptyset$.
- Now, in $\text{reverse}(B)$, $L_{s_1} \cap L_{s_2} = \emptyset$. 

Note that $A$ and $\text{reverse}(B)$ accept the same language. The catch is, $\text{reverse}(B)$ may not be a DFA. If we run subset construction on $\text{reverse}(B)$, then $L_{s_1} \neq L_{s_2}$ for any pair of states in the resulting DFA. Thus the resulting DFA will be minimal and equivalent to $A$. 

Compiler Design

Languages and Minimality

Partition Refinement

Other Minimization Strategies
Brzozowski’s Algorithm: Why it works

- Let $A$ be an NFA/DFA;
  Let $B$ be an automaton generated by the subset construction algorithm (NFA to DFA) of $\text{reverse}(A)$.

- Let $s_1$ and $s_2$ be two distinct states in $B$.
  Then, $L_{s_1} \cap L_{s_2} = \emptyset$.

- Now, in $\text{reverse}(B)$, $L_{s_1} \cap L_{s_2} = \emptyset$.

- Note that $A$ and $\text{reverse}(B)$ accept the same language.
  The catch is, $\text{reverse}(B)$ may not be a DFA.
Brzozowski’s Algorithm: Why it works

- Let $A$ be an NFA/DFA;
  Let $B$ be an automaton generated by the subset construction algorithm (NFA to DFA) of $\text{reverse}(A)$.
- Let $s_1$ and $s_2$ be two distinct states in $B$.
  Then, $L_{\rightarrow s_1} \cap L_{\rightarrow s_2} = \emptyset$.
- Now, in $\text{reverse}(B)$, $L_{s_1\rightarrow} \cap L_{s_2\rightarrow} = \emptyset$.
- Note that $A$ and $\text{reverse}(B)$ accept the same language.
  The catch is, $\text{reverse}(B)$ may not be a DFA.
- If we run subset construction on $\text{reverse}(B)$, then $L_{s_1\rightarrow} \neq L_{s_2\rightarrow}$ for any pair of states in the resulting DFA.
Brzozowski’s Algorithm: Why it works

- Let $A$ be an NFA/DFA;
  Let $B$ be an automaton generated by the subset construction algorithm (NFA to DFA) of $\text{reverse}(A)$.

- Let $s_1$ and $s_2$ be two distinct states in $B$.
  Then, $L_{s_1} \cap L_{s_2} = \emptyset$.

- Now, in $\text{reverse}(B)$, $L_{s_1} \cap L_{s_2} = \emptyset$.

- Note that $A$ and $\text{reverse}(B)$ accept the same language.
  The catch is, $\text{reverse}(B)$ may not be a DFA.

- If we run subset construction on $\text{reverse}(B)$, then $L_{s_1} \neq L_{s_2}$ for any pair of states in the resulting DFA.

- Thus the resulting DFA will be minimal and equivalent to $A$. 
Complexity Results

- Moore’s Algorithm runs in $O(n^2)$ time with appropriate data structures.
  (where $n =$ number of states of input DFA)
Complexity Results

- Moore’s Algorithm runs in $O(n^2)$ time with appropriate data structures.
  (where $n =$ number of states of input DFA)
- Brzozowski’s Algorithm is exponential in the worst case.
Complexity Results

- Moore’s Algorithm runs in $O(n^2)$ time with appropriate data structures.
  (where $n =$ number of states of input DFA)
- Brzozowski’s Algorithm is exponential in the worst case.
  Consider $L = (a|b)^k a(a|b)^*$: words where the $k + 1$-th symbol is an $a$. 

Complexity Results

- Moore’s Algorithm runs in $O(n^2)$ time with appropriate data structures.
  (where $n =$ number of states of input DFA)
- Brzozowski’s Algorithm is exponential in the worst case.
  - Consider $L = (a|b)^k a(a|b)^*$: words where the $k + 1$-th symbol is an $a$.
  - $L$ can be recognized by an automaton of size $O(k)$.
Complexity Results

- Moore’s Algorithm runs in $O(n^2)$ time with appropriate data structures.
  (where $n =$ number of states of input DFA)
- Brzozowski’s Algorithm is exponential in the worst case.
  - Consider $L = (a|b)^k a(a|b)^*$: words where the $k + 1$-th symbol is an $a$.
  - $L$ can be recognized by an automaton of size $O(k)$.
  - Reverse of $L$: words where the $k + 1$-th symbol from the end is an $a$. 
Moore’s Algorithm runs in $O(n^2)$ time with appropriate data structures.

(Where $n = \text{number of states of input DFA}$)

Brzozowski’s Algorithm is exponential in the worst case.

- Consider $L = (a|b)^k a(a|b)^*$: words where the $k + 1$-th symbol is an $a$.
- $L$ can be recognized by an automaton of size $O(k)$.
- Reverse of $L$: words where the $k + 1$-th symbol from the end is an $a$.
- Smallest DFA that can recognize $\text{reverse}(L)$ is of size $\Omega(2^k)$.
Complexity Results

- Moore’s Algorithm runs in $O(n^2)$ time with appropriate data structures. (where $n =$ number of states of input DFA)
- Brzozowski’s Algorithm is exponential in the worst case.
  - Consider $L = (a|b)^k a(a|b)^*$: words where the $k + 1$-th symbol is an $a$.
  - $L$ can be recognized by an automaton of size $O(k)$.
  - Reverse of $L$: words where the $k + 1$-th symbol from the end is an $a$.
  - Smallest DFA that can recognize $\text{reverse}(L)$ is of size $\Omega(2^k)$.
- Hopcroft’s algorithm maintains a waiting set of splitters, and can be done in $O(n \log n)$ time.