NFA to DFA via Subset Construction

- $\epsilon$-closure
- goto
- final states
Languages and Minimality

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 $\mathcal{L}_A$ 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)

- $L_{s0\rightarrow} = (ab)^+$
- $L_{s1\rightarrow} = b(ab)^*$
- $L_{s2\rightarrow} = (ab)^*$
- $L_{\rightarrow s1} = a(ba)^*$
- $L_{\rightarrow s2} = (ab)^+$
Languages and Minimality

Examples of Suffix and Prefix Languages (2)

The following automaton is not minimal:

\[ L_{s_1} = (a|b)^* \]
\[ L_{s_2} = (a|b)^* \]
\[ L_{s_0} = a(a|b)^* \]
\[ L_{s_2} = (a+)b(a|b)^* \]

The following automaton is minimal:

\[ L_{s_0} = (ab)^+ \]
\[ L_{s_1} = b(ab)^* \]
\[ L_{s_2} = (ab)^* \]

Languages and Minimality

Equivalence and Minimality (1)

- An automaton is *minimal* if the suffix languages of all states are pairwise distinct.
- The following automaton is not minimal:

\[ L_{s_1} = L_{s_2} = (a|b)^* \]

- The following automaton is minimal:

\[ L_{s_0} = (ab)^+ \]
Languages and Minimality

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:
```

Minimized automaton:

```
Minimized automaton:
```

Partition Refinement

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.
**Partition Refinement**

**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 \( P \).
- 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 \( P_{i+1} = P_i \land \bigwedge_{\alpha \in \Sigma} \alpha^{-1}.P_i \)
- Repeat until \( P_{n+1} = P_n \)
Another Example of Moore’s Algorithm

- $P_0$: $s_0, a, b \rightarrow s_1, a, b \rightarrow s_2, a, s_3$

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

- $P_1$: $s_0, a, b \rightarrow s_1, a, b \rightarrow s_2, a, s_3$

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

- $P_2$: $s_0, a, b \rightarrow s_1, a, b \rightarrow s_2, a, s_3$

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 \( \text{reverse}(A) \).
- 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)
- 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.