Equivalences of Finite Automata

Antonina Nepeivoda
`a_nevod@mail.ru`

Machines Comparison
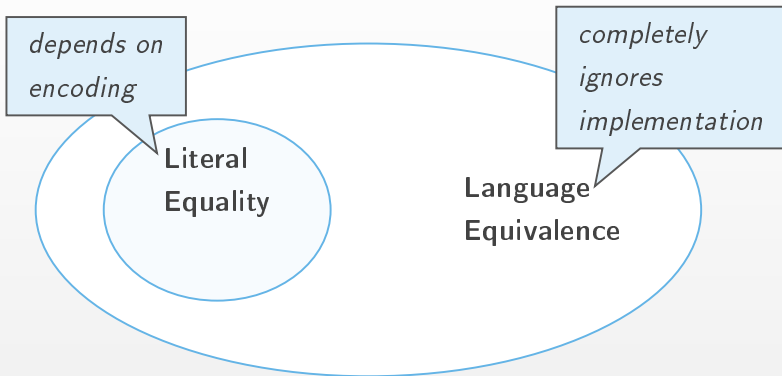
Problem: *how to find an equivalence that is sustainable to irrelevant implementation details (such as node naming) but tracks parsing-relevant properties?*

Equality of DFA and NFA

Given a DFA, its states can be *canonically named*: i.e. state number is determined by the string marking the shortest path from the starting state to the state considered. Then the numeration depends only on the chosen linear order on strings.
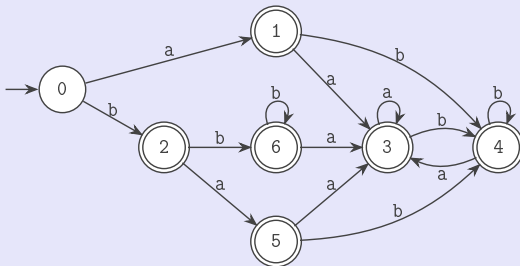
Above is the state numeration with respect to the military (length- lexicographic) order, given a ≺ b:
$\varepsilon \prec a \prec b \prec aa \prec ab \prec ba \prec bb$.

Equality of DFA and NFA

Given a DFA, its states can be *canonically named*: i.e. state number is
determined by the string marking the shortest path from the starting
state to the state considered. Then the numeration depends only on the
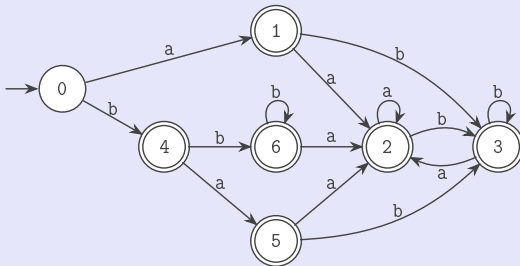chosen linear order on strings.

*Another numeration is induced by the "vanilla" lexicographic
order, given* $a \prec b$: *now* $\varepsilon \prec a \prec aa \prec ab \prec b \prec ba \prec bb$.

Equality of DFA and NFA

Given a DFA, its states can be *canonically named*: i.e. state number is determined by the string marking the shortest path from the starting state to the state considered. Then the numeration depends only on the chosen linear order on strings.

However, the canonical numeration does not work for NFA, provided that the string sets marking paths to the states can coincide.

Hence, DFA equality up to the state renaming can be thought as the literal coincidence of the canonically ordered DFA; but the canonical order does not work for recognising NFA equality.

Behavioral Equivalence and Language Equivalence

Language equivalence tracks only admissible actions, but not a way the actions are performed. The following example is well-known.

- N is «put a note into the machine»;
- C is «order a coffee»;
- T is «order a tea».
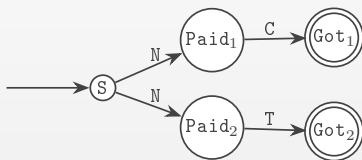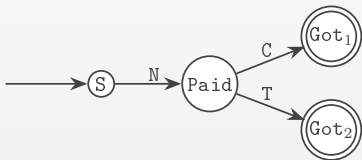
The given two machines have the same language, but are distinct from the point of view of a user:

- the first requires a note, and then asks what drink is ordered;
- the second asks for a choice when taking money, then requires to press the button that prepares it.

Bisimulation of Labelled Transition Systems

Bisimulation is a relation ~ between states of the systems $\mathcal{T}_1$ and $\mathcal{T}_2$ satisfying the following property:

- *If $q_1 \sim q_2$ ($q_1 \in \mathcal{T}_1$, $q_2 \in \mathcal{T}_2$), then for every transition $q_1 \xrightarrow{\gamma} q_1'$ in $\mathcal{T}_1$ there is a transition $q_2 \xrightarrow{\gamma} q_2'$ in $\mathcal{T}_2$ such that $q_1' \sim q_2'$, and vice versa.*

Starting and final (if any) states must be bisimilar.

Every state machine $\mathscr{A}$ can be represented as a labelled transition system.
- $\mathscr{A}_1$ and $\mathscr{A}_2$ are bisimilar $\Leftrightarrow$ their LTS $\mathcal{T}_1$ and $\mathcal{T}_2$ are bisimilar.

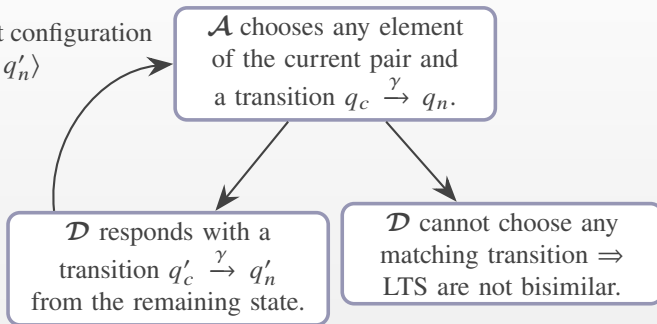Labelled transition systems are not necessarily finite; moreover, LTS contain no final states.

Existence of final states can be modelled via introducing *endmarkers* (usually denoted $) . Then every final state of an NFA has a transition by the endmarker to the unique «bottom» state.

Bisimulation Game

$\mathcal{T}_1$ and $\mathcal{T}_2$ bisimilarity checking technique can be formulated as a two-player game with an initial configuration $\langle q_S, q'_S \rangle$:

Next configuration $\langle q_n, q'_n \rangle$

$\mathcal{A}$ chooses any element of the current pair and a transition $q_c \xrightarrow{\gamma} q_n$.

$\mathcal{D}$ responds with a transition $q'_c \xrightarrow{\gamma} q'_n$ from the remaining state.

$\mathcal{D}$ cannot choose any matching transition $\Rightarrow$ LTS are not bisimilar.

- Attacker's winning strategy always leads to the fact that any possible play is finite.

- In presence of final states, $\mathcal{A}$ can additionally declare «game-over» in any final state (and $\mathcal{D}$ must respond with the «game-over» as well).

Equivalent Trim DFA are Bisimilar

Given two non-bisimilar trim DFA $\mathscr{A}_1$ and $\mathscr{A}_2$, we assume by the contradiction that the player $\mathcal{A}$ has a winning strategy. The strategy is completely determined by a finite input string ω.

- If after reading the string ω one DFA ends up in a final state, while the other ends up in a non-final state, then ω witnesses that their languages do not coincide.

- If after reading the string ω one DFA (say $\mathscr{A}_1$) ends up in a state with an outgoing transition by some $\gamma \in \Sigma$, while the other does not has such a transition, then no string in $\mathscr{L}(\mathscr{A}_1)$ with the prefix $\omega\gamma$ can belong to $\mathscr{L}(\mathscr{A}_2)$. The DFA are trim $\Rightarrow$ at least one string prefixed by $\omega\gamma$ is in $\mathscr{L}(\mathscr{A}_1)$.

Bisimulation and Equality

Bisimilar DFA are not necessarily equal, even if cardinalities of their state sets are also equal.

In this example, the distinction between the states c_1 and c_3 is redundant: they are indistinguishable with respect to the languages that can be recognised from them, namely, $\mathscr{L}(c_1) = \mathscr{L}(c_3) = \{\varepsilon\}$. Hence, $c_1 \sim c_3$.

We could *merge* the bisimilar states with no impact to the recognised DFA language; conversely, if we know that the DFA states coincide wrt their languages, then we know they are behaviorally equivalent.

k-Bisimulation: Playing Backwards

Given an NFA $\mathscr{A}$, how do we know that its states q_i, q_j are bisimilar?

- if q_i is final, while q_j is non-final, then $\mathcal{A}$ can choose q_i and declare game-over, winning the game. Hence, $\mathcal{A}$ can win doing no move at all. If q_i, q_j are both final or both non-final, at least one move is required for $\mathcal{A}$ to win, and we say that $q_i \sim_0 q_j$.

- if exists γ and q_i' s.t. $q_i \xrightarrow{\gamma} q_i'$, and for all q_j' s.t. $q_j \xrightarrow{\gamma} q_j'$ $q_i' \nsim_k q_j'$, then $\mathcal{A}$ wins in $k+1$ moves starting from the position $\langle q_i, q_j \rangle$. Otherwise, we say that $q_i \sim_{k+1} q_j$.

A closer look on the $\sim_{k+1}$-condition:

$$q_i \sim_{k+1} q_j \Leftrightarrow \forall q_i', \gamma \left(q_i \xrightarrow{\gamma} q_i' \Rightarrow \exists q_j' (q_j \xrightarrow{\gamma} q_j') \right)$$
$$\& \ \forall q_j', \gamma \left(q_j \xrightarrow{\gamma} q_j' \Rightarrow \exists q_i' (q_i \xrightarrow{\gamma} q_i') \right)$$

When we reach the fixpoint of $\sim_k$ (i.e. $\sim_k = \sim_{k+1}$), then we know that $\mathcal{A}$ never cannot win in position $\langle q_i, q_j \rangle$ given $q_i \sim_k q_j$, hence $q_i \sim q_j$.

Bisimilarity in DFA and Minimization

Given a DFA with state set Q, and set of final states F, we know that $q_i \sim q_j$ if and only if the sets of words recognised starting from q_i and starting from q_j are equal.

Hence, we can minimize the DFA by merging bisimilar states.

DFA Minimization Algorithm

---

1. Given $q_i, q_j \in Q$, mark the pairs $\{q_i, q_j\}$ s.t. $q_i \not\sim_0 q_j$ (i.e. $q_i \in F$, while $q_j \notin F$, or vice versa).

2. Mark all the pairs $\{q_i, q_j\}$ s.t.
   $\exists \gamma (q_i \xrightarrow{\gamma} q_i' \ \& \ q_j \xrightarrow{\gamma} q_j' \ \& \ \{q_i', q_j'\}$ is marked$)$.

3. Repeat Step 2 until no new marked pair appear (i.e. fixpoint of $\sim_k$ is reached).

4. Merge the states in the all unmarked pairs: they are bisimilar.

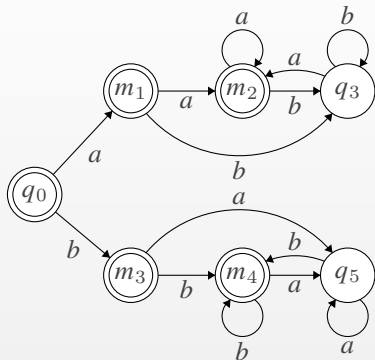# Minimisation Example

The initial DFA is:
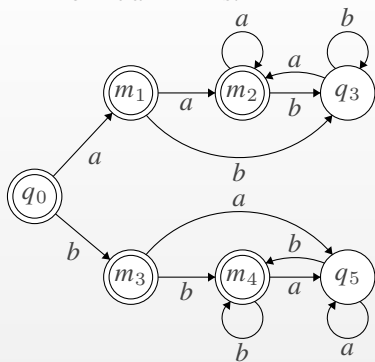
First, we construct the table in order to track marked pairs. We are required to track the pair only once, hence we can consider only the table part below its diagonal.
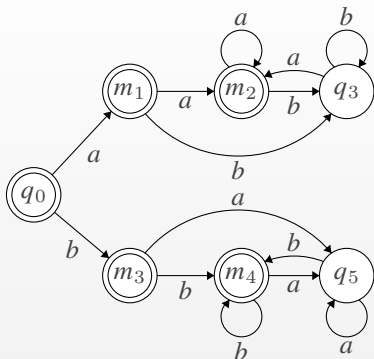
Minimisation Example

	q_0	m_1	m_2	q_3	m_3	m_4
m_1						
m_2						
q_3	$\checkmark$	$\checkmark$	$\checkmark$			
m_3				$\checkmark$		
m_4				$\checkmark$		
q_5	$\checkmark$	$\checkmark$	$\checkmark$		$\checkmark$	$\checkmark$

We have marked the states being not 0-bisimilar so far.

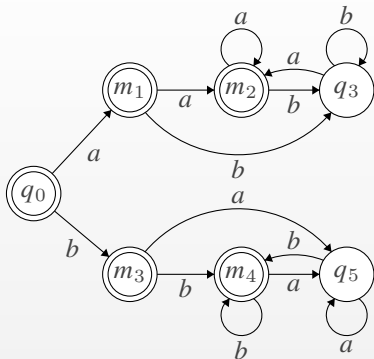# Minimisation Example

Now we are checking 1-bisimilarity:

$q_0 \xrightarrow{a} m_1, m_1 \xrightarrow{a} m_2$ $\quad q_0 \xrightarrow{b} m_3, m_1 \xrightarrow{b} q_3$ $\quad \{m_1, m_2\} \xrightarrow{a} m_2$ $\quad \{m_1, m_2\} \xrightarrow{b} q_3$

$q_0 \xrightarrow{a} m_1, m_2 \xrightarrow{a} m_2$ $\quad q_0 \xrightarrow{b} m_3, m_2 \xrightarrow{b} q_3$

The states q_0 and m_1 are distinguishable by b, as well as q_0 and m_2. The states m_1 and m_2 behave equally, i.e. we can say in advance that $m_1 \sim m_2$.

Minimisation Example
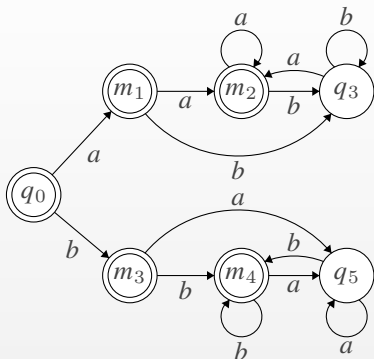

We continue checking 1-bisimilarity:

$q_0 \xrightarrow{a} m_1, m_3 \xrightarrow{a} q_5$ $q_0 \xrightarrow{a} m_1, m_4 \xrightarrow{a} q_5$ $m_1 \xrightarrow{a} m_2, m_3 \xrightarrow{a} q_5$

$m_2 \xrightarrow{a} m_2, m_3 \xrightarrow{a} q_5$ $m_1 \xrightarrow{a} m_2, m_4 \xrightarrow{a} q_5$ $m_2 \xrightarrow{a} m_2, m_4 \xrightarrow{a} q_5$

$q_3 \xrightarrow{a} m_2, q_5 \xrightarrow{a} q_5$

We find out that a-transitions distinguish almost all remaining pairs.

Minimisation Example

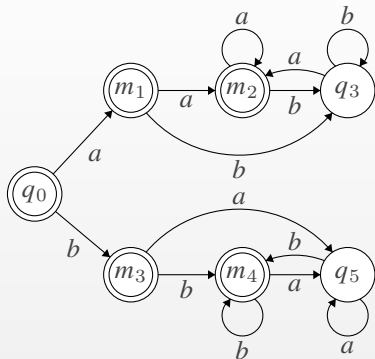|						
m_1	✓					
m_2	✓					
q_3	✓	✓	✓			
m_3	✓	✓	✓	✓		
m_4	✓	✓	✓	✓		
q_5	✓	✓	✓	✓	✓	✓
	q_0	m_1	m_2	q_3	m_3	m_4

The one more exception are the states m_3, m_4, behaving equally on both terminal letters.

$$\{m_3, m_4\} \xrightarrow{a} q_5 \qquad \{m_3, m_4\} \xrightarrow{b} m_4$$

We cannot distinguish m_3 and m_4, as well as m_1 and m_2, by 2-bisimilarity relation, hence, the relation fixpoint is reached.
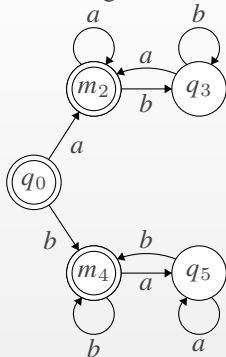
Minimisation Example

Now we can merge the pairs m_1, m_2 and m_3, m_4, constructing a DFA with less number of states.

Minimisation Example

The resulting DFA is:

m_1	$\checkmark$					
m_2	$\checkmark$					
q_3	$\checkmark$	$\checkmark$	$\checkmark$			
m_3	$\checkmark$	$\checkmark$	$\checkmark$	$\checkmark$		
m_4	$\checkmark$	$\checkmark$	$\checkmark$	$\checkmark$		
q_5	$\checkmark$	$\checkmark$	$\checkmark$	$\checkmark$	$\checkmark$	$\checkmark$
	q_0	m_1	m_2	q_3	m_3	m_4

Ok, we have reduced the DFA. But how we can guarantee that the resulted DFA is really minimal, i.e. there is no DFA with less number of states recognising the same language?