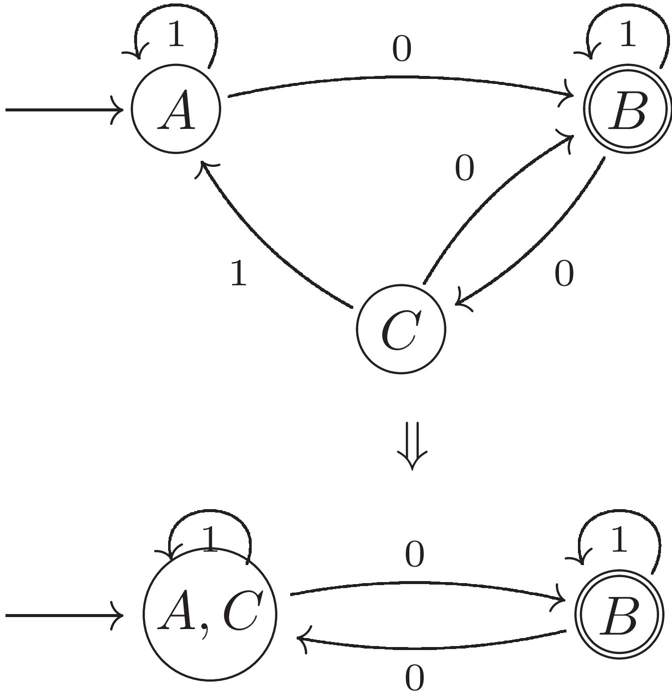
Griffith University

3515ICT Theory of Computation

Minimisation of DFAs

(Based loosely on slides by Harald Søndergaard of The University of Melbourne)

Example 1



Outline

1. The Myhill-Nerode theorem [1] gives yet *characterisation* of regular languages. It leads to a (non-constructive) definition of a *minimal* DFA that recognises a regular language L. A corollary is that the minimal DFA for L is *unique* (up to renaming).
2. A (distinguishability) algorithm from Hopcroft *et al.* [2] is an $O(|Q|^3)$ algorithm.
3. An (equivalence) algorithm from Aho *et al.* [3] is an $O(|Q|^2)$ (and more direct) algorithm.

Equivalence relations and partitions

Definition An *equivalence relation* is a binary relation $\equiv$ on a set S that is *reflexive* ($x \equiv x$), *symmetric* ($x \equiv y$ implies $y \equiv x$) and *transitive* ($x \equiv y$ and $y \equiv z$ implies $x \equiv z$).

Lemma Every equivence relation $\equiv$ on a set S partitions S into disjoint components (or equivalence classes) $C_1, C_2, \ldots$ such that all pairs of elements in each component are equivalent and no pair of elements in different components are equivalent.

Definition Let $\equiv$ be an equivalence relation on a set S. Let $x \in S$. Then $[x]_\equiv$ is the component of the partition for $\equiv$ containing x.

Myhill-Nerode Theorem

Let L be any language over a finite alphabet Σ.

Definition Let $\equiv_L$ be the equivalence relation on Σ^* such that $s \equiv_L t$ if and only if, for all $w \in \Sigma^*$, $sw \in L$ iff $tw \in L$.

Example 1 Let $L = \{\, 0^n 1^n \mid n \geq 0 \,\}$. Then the components of the partition for $\equiv_L$ are
$\{\, 0^{m+k} 1^m \mid m \geq 0 \,\}$ for each $k \geq 0$, and
$\{\, s \in \{0, 1\}^* \mid s$ is not a prefix of a string in $L \,\}$.

Example 2 Let L be the set of strings over Σ^* representing binary integers evenly divisible by 3. Then the three components of the partition for $\equiv_L$ are the sets of strings representing binary integers divisible by 3 with remainders 0, 1 and 2, respectively.

Myhill-Nerode (cont.)

Theorem L is regular if and only if $\equiv_L$ partitions Σ^* into a *finite* number of components.

The Myhill-Nerode theorem provides an alternative way to prove a language is not regular:

Let L be a language over Σ. Let $\equiv_L$ be the equivalence relation on Σ^* determined by L. Then L is not regular iff $\equiv_L$ partitions Σ^* into an infinite number of components.

Hence, Example 1 on the previous slide is not regular.

Exercise Let P be the language of palindromes over $\{a, b\}$. Construct the components of the partition for $\equiv_P$. Hence show that P is not regular.

Myhill-Nerode (cont.)

Let L be a regular language over alphabet Σ.

Let $M_L = (Q, \Sigma, \delta, q_0, F)$, where Q is the set of components for $\equiv_L$, $q_0 = [\epsilon]_{\equiv_L}$, $F = \{ [s]_{\equiv_L} \mid s \in L \}$, and $\delta([s]_{\equiv_L}, a) = [sa]_{\equiv_L}$, for all $s \in \Sigma^*$ and $a \in \Sigma$.

Then M_L is a DFA that recognises L!

In fact, M_L is the *minimal* DFA that recognises L.

Finally, every minimal DFA that recognises L is *identical* to M_L (up to state renaming). *I.e.*, there exists a *unique* minimal DFA that recognises L (up to state renaming).

But this definition is *very* nonconstructive...

Algorithm 1: Distinguishability

Ref: Hopcroft *et al.* [2]; Martin [4].

From now on, assume all DFAs have no unreachable states.

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA for a regular language L.

Definition Two states $p, q \in Q$ are *distinguishable* if there exists a string $w \in \Sigma^*$ such that $\delta^*(p, w) \in F$ iff $\delta^*(q, w) \notin F$. Two states that are not distinguishable are called *equivalent*.

Lemma A DFA is minimal iff every pair of states is distinguishable.

Informally, Algorithm 1 below transforms M into a minimal DFA by merging all pairs of equivalent states into a single state.

Algorithm 1 (cont.)

1. (Mark all pairs of distinguishable states.)

   First, mark all pairs p, q, where $p \in F$ and $q \notin F$, as distinguishable. Then,

   repeat
     for all unmarked pairs p, q **do**
       for each $a \in \Sigma$ do
         if the pair $\delta(p, a)$, $\delta(q, a)$ is marked
         then mark the pair p, q
   until no new pairs are marked.

2. (Mark all pairs of states not marked distinguishable as equivalent.)

   If two states p, q are not marked as distinguishable, mark them as equivalent. This defines an equivalence relation $\equiv$ on Q.

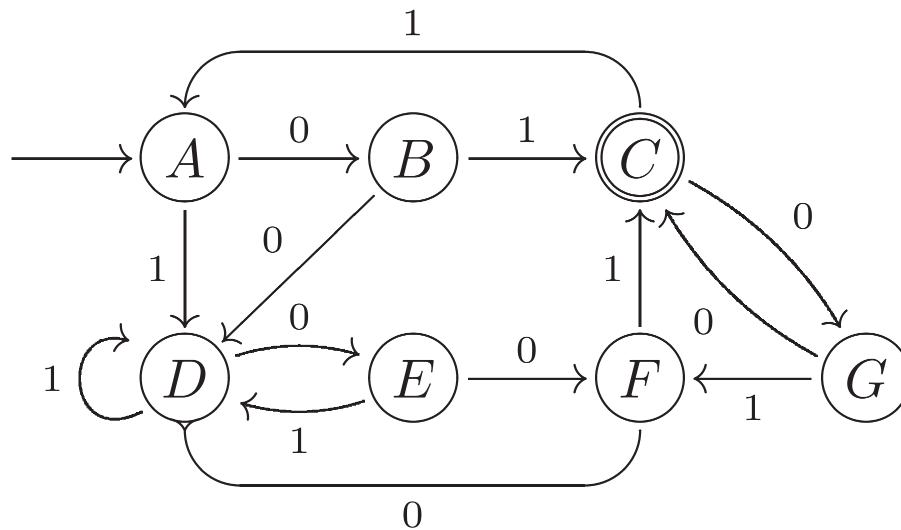
Algorithm 1 (cont.)

3. (Construct the minimal DFA for L from the equivalence classes for $\equiv$.)

   Let $M_L = (Q', \Sigma, \delta', q_0', F')$, where
   $Q' = \{ [q]_\equiv \mid q \in Q \}$, $q_0' = [q_0]_\equiv$,
   $F' = \{ [q]_\equiv \mid q \in F \}$, and $\delta'([p]_\equiv, a) = [q]_\equiv$,
   where $\delta(p, a) = q$, for each $p \in Q$ and $a \in \Sigma$.

This algorithm has complexity $O(|\Sigma| \times |Q|^3)$.

Example 2



Marking distinguishable pairs of states, we first mark the pairs (A,C), (B,C), (C,D), (C,E), (C,F) and (C,G).

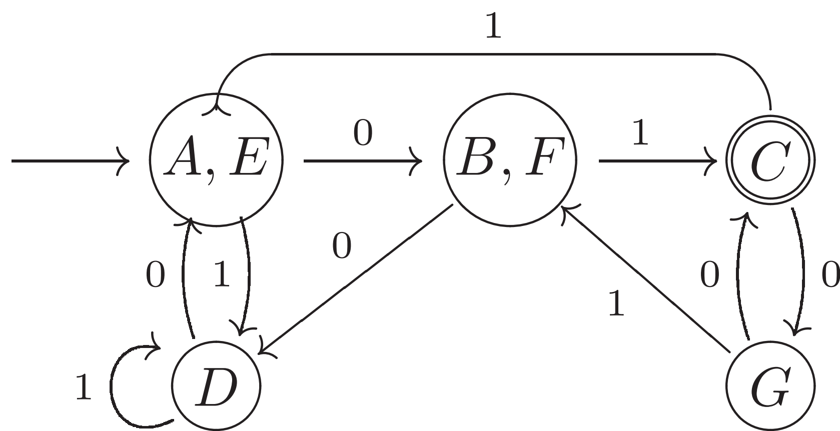
We then mark (A,B), (A,F), (B,D), (B,E), (B,G), (D,F), (E,F), (E,G) and (F,G).

We next mark (A,D), (A,G), (B,G) and (D,G), but that's all.

So, the equivalent pairs of states are (A,E) and (B,F).

Example 2 (cont.)

Merging equivalent states gives:



This DFA is minimal, since every pair of states is distinguishable.

Algorithm 2: Equivalence

Ref: Aho *et al.* [3].

Algorithm 1 is intuitively inefficient since it first computes distinguishable pairs, then equivalent pairs, and finally equivalence classes, when all we really want are the equivalence classes themselves.

The following algorithm computes the equivalence classes directly. This makes the algorithm both simpler and faster.

Algorithm 2 (cont.)

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA for a regular language L as above.

For every partition P of Q, define $\mathit{splits}(C_0, C_1, a) \equiv \exists\, p, q \in C_0$ s.t. $\delta(p, a) \in C_1$ and $\delta(q, a) \notin C_1$, for $C_0, C_1 \in P$ and $a \in \Sigma$. (We could say "a splits C_0 using C_1".)

$P := \{Q \setminus F, F\}$
while $\exists\, C_0, C_1 \in P,\ a \in \Sigma$ s.t.$\mathit{splits}(C_0, C_1, a)$ **do**
  let $C_0, C_1 \in P,\ a \in \Sigma$ s.t $\mathit{splits}(C_0, C_1, a)$
  $C_0' := \{\, p \in C_0 \mid \delta(p, a) \in C_1 \,\}$
  $P := P \setminus \{C_0\} \cup \{C_0', C_0 \setminus C_0'\}$
end while

(Of course, in practice, we don't repeat the search for $C_0, C_1 \in P,\ a \in \Sigma$ s.t. $\mathit{splits}(C_0, C_1, a)$.)

Algorithm 2 (cont.)

Now, the partition P defines an equivalence relation on Q by $p \equiv q$ iff p and q belong to the same component of P.

We can then construct the mininal DFA for L from the components of P as in Algorithm 1.

The complexity of Algorithm 2 is $O(|\Sigma| \times |Q|^2)$, using an efficient set representation.

Example 3

Ref: Hopcroft *et al.* [2], Fig. 4.8.

	0	1
→ A	B	F
B	G	C
*C	A	C
D	C	G
E	H	F
F	C	G
G	G	E
H	G	C

5-state DFA

Example 4

Ref: Martin [4], Fig. 5.3(a).

	0	1
$\rightarrow 1$	2	3
2	4	5
3	6	7
4	4	5
5	6	7
*6	4	5
7	6	7

3-state DFA

Equivalence of regular languages

These techniques provide an efficient way to test the equality of regular languages.

Two DFAs recognise the same language iff their minimal DFAs are identical (up to state renaming).

Even more efficiently, two DFAs recognise the same language iff their initial states are equivalent after taking the union of the two DFAs.

(This may avoid the complete computation of the two minimal DFAs.)

References

1. M. Sipser, *Introduction to the Theory of Computation, Second Edition.* Problems 1.51–1.52.

2. J.E. Hopcroft *et al.*, *Introduction to Automata Theory, Languages, and Computation, Second Edition.* Section 4.4.

3. A.V. Aho *et al.*, *Compilers: Principles, Techniques, and Tools.* Algorithm 3.6.

4. J. Martin, *Introduction to Languages and the Theory of Computation, Third Edition.* Sections 5.1–5.2.