TDDD14/TDDD85 Lecture 5: Minimization of Automata

Victor Lagerkvist (based on slides by Christer Bäckström)

In this lecture we describe a *minimisation* algorithm which for a given DFA produces an equivalent DFA with a minimal number of states. This procedure can be combined with earlier algorithms and can e.g. be used to compile a regular expression to a DFA with a minimal number of states.

Background and Intuition

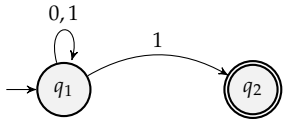

Figure 1: An NFA for $\{x1 \mid x \in \{0,1\}^*\}$.

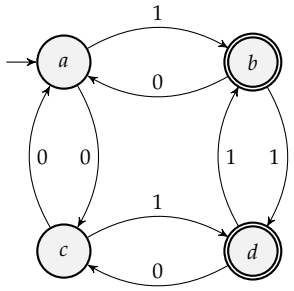

Figure 2: A DFA for the language $\{x1 \mid x \in \{0,1\}^*\}$.

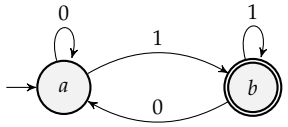

Figure 3: A smaller DFA accepting $\{x1 \mid x \in \{0,1\}^*\}$.

By now it should not come as a surprise that not all automata are created equal, and that one for each regular language can define many different DFAs which recognises the language. Among these infinite possibilities, should there be a preference? If no other heuristic is given then it is reasonable that we prefer a DFA which is as small and simple as possible, since this (1) typically makes it easier to understand the automaton, and (2) a DFA with fewer states can be simulated more efficiently. For example, when we studied regular expressions in the previous lecture we described a method for compiling regular expressions to DFA by first converting the regular expression to an NFA, and then applying the subset construction from lecture 3. Unfortunately, the subset construction typically fails to produce DFAs that are minimal with respect to the number of states, even when much smaller DFAs exist. For example, consider the NFA in Figure 1 which recognises the language $\{x1 \mid x \in \{0,1\}^*\}$. If we apply the subset construction to this automaton then we obtain a DFA with 4 states, one state for every non-empty subset of $\{q_1, q_2\}$. This DFA is visualized in Figure 2, where we have taken the liberty of renaming the 4 states to a, b, c, d. By previous lectures we already know that a smaller DFA exists (Figure 3), but can we somehow construct the DFA with only 2 states from the larger DFA with 4 states? Hence, we want a *minimisation* algorithm which given a DFA constructs an equivalent DFA with a minimal number of states.

Consider the following concerning the DFA in Figure 2: regardless of whether we are in state a, b, c, d and have just read the symbol 1, then we are in state b or d, which are both accept states. Similarly, if we have just read the symbol 0, then we must be in state a or c, and neither is an accept state. Hence, it does not matter whether we are in state b or state d since (1) both are accept states, (2) reading 1 still results in an accept state, and (3) reading 0 results in a non-accept state. Similarly, the machine does not need to separate between a and

c since reading 1 results in an accept state, and reading 0 results in a non-accept state. Hence, in a certain sense, b is *equivalent* with d, and a is equivalent with c. This suggests that we can merge a and c to a single state, and b and d to a single state, and update all transitions accordingly (Figure 3). We will see that this idea is more general and that it is possible to define an *equivalence relation* over the set of states of a DFA which can be used to define a state-minimal *quotient automaton*. We use these ideas and propose a minimisation algorithm which given a DFA computes an equivalent quotient automaton by iteratively computing all states that are equivalent and can be collapsed into a single state.

Equivalence Relations

Before turning to the concrete case of DFA we recall the important concept of an equivalence relation.

Definition 1. *A binary relation R on a set S is an equivalence relation if it satisfies the following three properties:*

1. *reflexive: $R(x,x)$ for all $x \in S$,*
2. *symmetric: $R(x,y) \Rightarrow R(y,x)$ for all $x,y \in S$,*
3. *transitive: $R(x,y)$ and $R(y,z) \Rightarrow R(x,z)$ for all $x,y,z \in S$.*

Example 1. *Let $\Sigma = \{0,1\}$. Define the binary relation R on Σ^* such that $R(x,y)$ if and only if $|x| = |y|$, i.e., two strings are considered equivalent if they have the same length. We claim that R is an equivalence relation over Σ^*, and to prove this claim we have to show that the three properties defining an equivalence relation hold for the relation R. This turns out to be very simple for R, but we spell out the details just to be sure.*

1. *For every $x \in \Sigma^*$ it holds that $|x| = |x|$. Hence, $R(x,x)$ and R is* **reflexive**.
2. *For all $x,y \in \Sigma^*$, if $|x| = |y|$, then $|y| = |x|$. Hence, $R(x,y)$ implies $R(y,x)$, and R is* **symmetric**.
3. *For all $x,y,z \in \Sigma^*$, if $|x| = |y|$ and $|y| = |z|$, then $|x| = |z|$. Hence, if $R(x,y)$ and $R(y,z)$, then $R(x,z)$, and R is* **transitive**.

The main point of an equivalence relation R is to be able to group elements together when they are considered to be "equivalent" with respect to R.

Definition 2. *Let R be an equivalence relation. Each string $x \in \Sigma^*$ has an associated* equivalence class *$[x]$, defined as $[x] = \{y \in \Sigma^* \mid R(x,y)\}$.*

Example 2. *We continue the previous example where $R(x, y)$ if and only if $|x| = |y|$. Then $[x]$ is the set of all strings that have the same length as x, including x itself. For example, we have*

- $[\varepsilon] = \{\varepsilon\}$ *(since ε is the unique string of length 0),*
- $[0] = [1] = \{0, 1\}$,
- $[00] = [01] = [10] = [11] = \{00, 01, 10, 11\}$,

and so on. Note that we have an infinite number of equivalence classes in this case since each string is related to the finite set of strings of the same length.

It follows from the definition that each element belongs to exactly one equivalence class. Let S be a set and R an equivalence relation on S. Let P be the set of all equivalence classes for R. Then P is a *partition* of S, i.e.:

- each equivalence class is non-empty,
- P covers S, i.e. every $x \in S$ belongs to some equivalence class, and
- if X and Y are equivalence classes s.t. $X \neq Y$, then $X \cap Y = \emptyset$.

In the previous example, R gives a partition with one equivalence class P_i for each $i \in \mathbb{N}$, such that $P_i = \{|x| \in \Sigma^* \mid |x| = i\}$. For instance, $P_0 = [\varepsilon]$ and $P_3 = [001]$.

Quotient Automata

Is it then possible to define an equivalence relation over the states of an automaton, so that two states which are deemed equivalent can be collapsed without affecting the language of the automaton? Hence, let $(Q, \Sigma, \delta, q_0, F)$ be a DFA, and let $p, q \in Q$ be two states. We begin by making the following two observations.

1. We cannot collapse p and q if $p \in F$ and $q \notin F$[1].
2. If we collapse p and q and there is some $a \in \Sigma$ such that $\delta(p, a) \neq \delta(q, a)$, then we must collapse also $\delta(p, a)$ and $\delta(q, a)$ to one state[2].

[1] Otherwise we cannot distinguish between accept and reject.

[2] Otherwise we have two choices on the symbol a.

More generally, the following extension of a transition function δ will be very useful to define the necessary equivalence relation.

Definition 3. *Let $(Q, \Sigma, \delta, q_0, F)$ be a DFA. Let $\hat{\delta}$ be the extension of δ to strings, defined such that for all states $p \in Q$:*

- $\hat{\delta}(p, \varepsilon) = p$, *and*

- $\hat{\delta}(p, xa) = \delta(\hat{\delta}(p, x), a)$ *for all* $x \in \Sigma^*$ *and all* $a \in \Sigma$.

Example 3. *For the DFA in Figure 2, restated in Figure 4, we e.g. have that*

1. $\hat{\delta}(a, \varepsilon) = a$,
2. $\hat{\delta}(a, 00) = a$,
3. $\hat{\delta}(a, 000) = c$,
4. $\hat{\delta}(a, 001) = b$,
5. $\hat{\delta}(b, 0011) = b$,
6. $\hat{\delta}(b, 00111) = d$, *and*
7. $\hat{\delta}(d, 001) = b$.

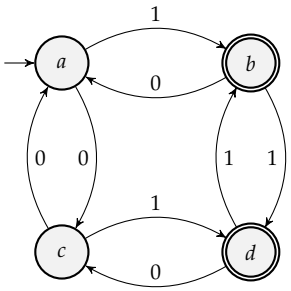

Figure 4: The DFA from Figure 2.

More generally these examples lead to the following observations. Regardless of our current state, if we have just read 0*, then we must be in a or c, and are therefore not in an accept state. Similarly, regardless of our current state, if we have just read* 1*, then we must be in b or d, i.e., an accept state. This leads to the following description of* $\hat{\delta}$ *with respect to the states a and c, for any* $x \in \Sigma^*$*:*

- $\hat{\delta}(a, x0) \notin F$ *and* $\hat{\delta}(c, x0) \notin F$, *and*
- $\hat{\delta}(a, x1) \in F$ *and* $\hat{\delta}(c, x1) \in F$.

Hence, for any *string* $x \in \Sigma^*$ *we have that* $\hat{\delta}(a, x) \in F$ *if and only if* $\hat{\delta}(c, x) \in F$. *Similarly, for any string* $x \in \Sigma^*$ *we also have that* $\hat{\delta}(b, x) \in F$ *if and only if* $\hat{\delta}(d, x) \in F$.

With the help of $\hat{\delta}$ we are now ready to define an equivalence relation over the states of an automaton.

Definition 4. *Let* $(Q, \Sigma, \delta, q_0, F)$ *be a DFA. Define the binary relation* $\approx$ *on the set* Q *of states such that for any two states* $p, q \in Q$

$$p \approx q$$

holds if and only if for all $x \in \Sigma^*$

$$\hat{\delta}(p, x) \in F \Leftrightarrow \hat{\delta}(q, x) \in F.$$

In other words two states p and q are treated as equivalent if the automaton when starting in either p or q on *any* input string reaches an accept state from p if and only if it reaches an accept state from q. Hence, when it comes to acceptance, which is all that matters in a DFA, the machine is unable to distinguish between the two states p and q.

Example 4. *In Example 3 we have already proven that $\hat{\delta}(a, x) \in F$ if and only if $\hat{\delta}(c, x) \in F$ for any string $x \in \Sigma^*$, and that $\hat{\delta}(b, x) \in F$ if and only if $\hat{\delta}(d, x) \in F$ for any string $x \in \Sigma^*$. But this is precisely the definition of the relation $\approx$, meaning that $a \approx c$ and $b \approx d$.*

We can then easily prove that $\approx$ results in an equivalence relation over the states of an automaton.

Lemma 1. *Let $(Q, \Sigma, \delta, q_0, F)$ be a DFA. Then $\approx$ is an equivalence relation over Q.*

Proof. The relation $\approx$ has the properties:

1. $p \approx p$ for all p (reflexive),
2. if $p \approx q$, then $q \approx p$ (symmetric), and
3. if $p \approx q$ and $q \approx r$, then $p \approx r$ (transitive).

Hence, $\approx$ is an equivalence relation. □

Furthermore, this defines an equivalence class $[p]$ for every state p as

$$[p] = \{q \mid q \approx p\}.$$

Recall that an equivalence relation defines a partition, so every state belong to exactly one equivalence class, i.e. $p \approx q$ if and only if $[p] = [q]$.

The idea is then to use the equivalence relation $\approx$ and construct an automaton containing one state for each equivalence class of $\approx$. This formalises the earlier intuition that we could collapse "equivalent" states.

Definition 5. *Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA. We define the* quotient automaton $M_{/\approx} = (Q', \Sigma, \delta', q_0', F')$ *where*

- $Q' = \{[p] \mid p \in Q\}$,
- $\delta'([p], a) = [\delta(p, a)]$,
- $q_0' = [q_0]$, *and*
- $F' = \{[p] \mid p \in F\}$.

Example 5. *Recall the automaton from Figure 4 where we in Example 4 proved that $a \approx c$ and $b \approx d$.*

This leads to the equivalence classes $[a]$ and $[b]$, and the quotient automaton $M_{/\approx}$ where $Q' = \{[a], [b]\}$ (Figure 5). Up to renaming of states, this is precisely the 2-state DFA from Figure 3, but we were able to construct it automatically once we had determined the equivelence relation $\approx$.

Trivia: the idea of grouping elements together according to an equivalence relation is a very powerful mathematical idea. In abstract and universal algebra this is known as a *quotient algebra*, and generalises the idea of a quotient automaton. For example, if we group together numbers which are congruent modulo 2, i.e., all even numbers, then addition and multiplication of these numbers remain even, and we obtain a well-defined quotient algebra.

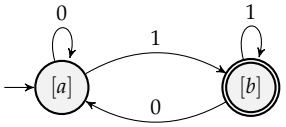

Figure 5: The minimised quotient automaton.

The idea behind the minimisation algorithm is then to compute the quotient automaton by iteratively computing the equivalence classes of $\approx$. We will see how this can be done in the next section, but before we turn to this we have to prove that the quotient automaton $M_{/\approx}$ actually defines the same language as the original automaton M (otherwise, the whole idea fails).

Theorem 1. *$L(M) = L(M_{/\approx})$ for any DFA M.*

Proof. Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA and let $M_{/\approx} = (Q', \Sigma, \delta', q_0', F')$ be its corresponding quotient automaton. Our aim is to prove that $L(M) = L(M_{/\approx})$, which we will accomplish by proving that for *every* input string x, M accepts the string if and only if $L(M_{/\approx})$ accepts the string. In symbols, this means that we need to prove that for any $x \in \Sigma^*$ it holds that $\hat{\delta}(q_0, x) \in F$ if and only if $\hat{\delta}'([q_0], x) \in F'$.

Hence, let $p \in Q$ and let $x \in \Sigma^*$ be an arbitrary string. We will prove by induction over the length of x that $\hat{\delta}(p, x) \in F$ if and only if $\hat{\delta}'([p], x) \in F'$.

So in particular this holds when $p = q_0$.

Base case: $|x| = 0$, so $x = \varepsilon$ is the only possibility.

- We have $\hat{\delta}(p, \varepsilon) = p$ and $\hat{\delta}'([p], \varepsilon) = [p]$.
- We have $p \in F$ if and only if $[p] \in F'$ by definition of $M_{/\approx}$.
- Hence, $\hat{\delta}(p, \varepsilon) \in F$ if and only if $\hat{\delta}'([p], \varepsilon) \in F'$.

Induction step: suppose the claim holds for all strings of length n, for some $n \geq 0$. We must prove that it holds also for strings of length $n + 1$.

- Let $a \in \Sigma$ and $x \in \Sigma^n$. Then $|ax| = n + 1$.
- Let $q = \delta(p, a)$.
- Then $\delta'([p], a) = [\delta(p, a)] = [q]$.
- It follows from the induction hypothesis that $\hat{\delta}(q, x) \in F$ if and only if $\hat{\delta}'([q], x) \in F'$.
- Hence, $\hat{\delta}(p, ax) \in F$ if and only if $\hat{\delta}'([p], ax) \in F'$.

This proves the claim, so it follows that for all $x \in \Sigma^*$, it holds that $\hat{\delta}(q_0, x) \in F$ if and only if $\hat{\delta}'([q_0], x) \in F'$. Hence, $L(M) = L(M_{/\approx})$. □

While the construction of a quotient automaton and the resulting proof of equivalence turned out to be more complicated than many of the arguments that we have seen earlier in the course, the basic idea is still (1) that some states can be proven to be equivalent when it comes to acceptance, and (2) to minimise the automaton

we collapse equivalent states into a single state. Hence, if you have difficulties understanding the formal proof of Theorem 1 then it is a good idea to return to it after having understood and applied the minimisation algorithm which we will now describe.

The Minimisation Algorithm

BASED ON the ideas in the preceding section we now describe an algorithm which for any given DFA M computes the quotient automaton $M_{/\approx}$ which describes the same language. We want to compute the equivalence relation $\approx$, and the idea is to gradually *mark* pairs of states which are *not* equivalent under $\approx$ and thus never can be collapsed into a single state. Initially, given two states p and q, what is the easiest possible check for this? This is just the case when p is an accept state, and q is not an accept state, since this implies that $p \notin [q]$. Hence, we perform this check for any pairs of states in the automaton, and mark these states. What should be the second check? Given states p and q we then check if there exist transitions from p and q (under the same input symbol) so that one of these transitions lead to a previously marked state, but the other to an unmarked state. In the next step, we repeat this again, but also take the marked states from the previous step into account. This idea can be generalised and summarised as follows.

Marking algorithm

1. For all pairs of states $\{p, q\}$:
   - if $p \in F$ and $q \notin F$, then mark $\{p, q\}$.
2. For all unmarked pairs of states $\{p, q\}$
   - if there is some $a \in \Sigma$ such that $\{\delta(p, a), \delta(q, a)\}$ is marked then mark $\{p, q\}$.
3. Repeat 2 until no new pair is marked.

If a pair of states $\{p, q\}$ is still unmarked, then $p \approx q$, and the minimised automaton is the quotient automaton $M_{/\approx}$. We consider a concrete example of the marking algorithm in the appendix on p. 9.

Food for Thought

1. We never actually prove that the marking algorithm and the resulting quotient automaton is the smallest (with respect to the number of states) automaton recognising the language in question.

Try to come up with a DFA where the quotient automaton is not the smallest possible one for the language in question. Hint: it is possible to come up with an example where the minimal automaton consists of only one state.

2. What is the time complexity of the proposed minimisation algorithm (with respect to the number of states of the given DFA)? Can the number of steps be bounded by a polynomial?

3. Can the minimisation algorithm be directly generalised to NFAs? Why, or why not?

Appendix

We now consider a concrete, larger example of the minimisation algorithm. Assume that we are given the DFA M in Figure 6.

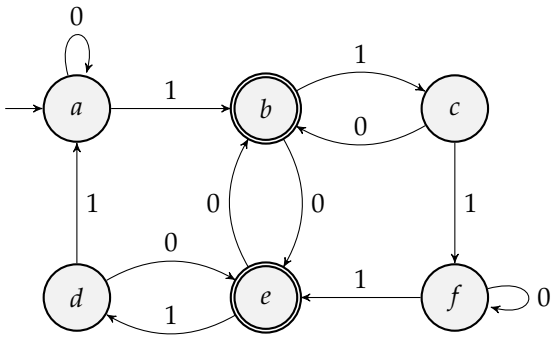

Figure 6: The DFA M.

We are interested in minimising the number of states and thus want to compute the equivalence relation $\approx$ so that we can compute the quotient automaton $M_{/\approx}$.

Iteration 0

We start off simple and draw the table in Figure 7.

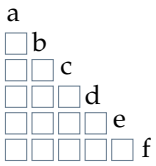

Figure 7: The initial table.

For each pair of states $\{p, q\}$ there now exists an entry in this table. Note that since the relation $\approx$ is symmetric we do not need one entry for a, b, and an additional entry for b, a, and so on. Next, we mark a pair of states $\{p, q\}$ if $p \in F$ and $q \notin F$ (Figure 8).

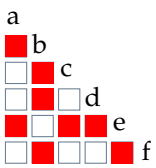

Figure 8: Iteration 0: mark an entry $\{p, q\}$ if $p \in F$ and $q \notin F$.

Iteration 1

For each pair $\{p, q\}$ which is *not* marked we now want to determine if there exists an input symbol $x \in \{0, 1\}$ such that $\{\delta(p, x), \delta(q, x)\}$

has already been marked, in which case we mark $\{p,q\}$ as well. In this particular iteration it simply means that we mark $\{p,q\}$ if there exists an input symbol and transitions from p and q such that one ends up in an accept state, but the other does not. This is easiest accomplished by doing it in a systematic fashion as follows.

1. $\{a,c\} : \{\delta(a,0),\delta(c,0)\} = \{a,b\}$ is marked, so mark $\{a,c\}$.
2. $\{a,d\} : \{\delta(a,0),\delta(d,0)\} = \{a,e\}$ is marked, so mark $\{a,d\}$.
3. $\{a,f\} : \{\delta(a,0),\delta(f,0)\} = \{a,f\}$ is unmarked, so also check 1.
4. $\{a,f\} : \{\delta(a,1),\delta(f,1)\} = \{b,e\}$ is unmarked, so do not mark.
5. $\{b,e\} : \{\delta(b,0),\delta(e,0)\} = \{b,e\}$ is unmarked, so also check 1.
6. $\{b,e\} : \{\delta(b,1),\delta(e,1)\} = \{c,d\}$ is unmarked, so do not mark.
7. $\{c,d\} : \{\delta(c,0),\delta(d,0)\} = \{b,e\}$ is unmarked, so also check 1.
8. $\{c,d\} : \{\delta(c,1),\delta(d,1)\} = \{f,a\}$ is unmarked, so do not mark.
9. $\{c,f\} : \{\delta(c,0),\delta(f,0)\} = \{b,f\}$ is marked, so mark $\{c,f\}$.
10. $\{d,f\} : \{\delta(d,0),\delta(f,0)\} = \{e,f\}$ is marked, so mark $\{d,f\}$.

This sequence of markings is visualized in Figure 9– 12 where any new marking is coloured in black. We thus end up with the markings in Figure 13.

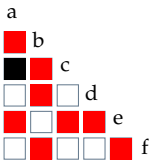

Figure 9: Marking $\{a,c\}$.

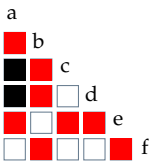

Figure 10: Marking $\{a,d\}$.

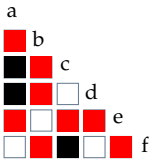

Figure 11: Marking $\{c,f\}$.

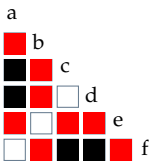

Figure 12: Marking $\{d,f\}$.

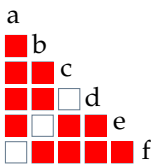

Figure 13: Iteration 1 (finished).

Iteration 2

Again, we repeat the step of finding an unmarked pair $\{p,q\}$ where $\{\delta(p,x),\delta(q,x)\}$ has already been marked, following exactly the same method as in iteration 1.

1. $\{a,f\} : \{\delta(a,0),\delta(f,0)\} = \{a,f\}$ is unmarked, so also check 1.
2. $\{a,f\} : \{\delta(a,1),\delta(f,1)\} = \{b,e\}$ is unmarked, so do not mark.
3. $\{b,e\} : \{\delta(b,0),\delta(e,0)\} = \{b,e\}$ is unmarked, so also check 1.
4. $\{b,e\} : \{\delta(b,1),\delta(e,1)\} = \{c,d\}$ is unmarked, so do not mark.
5. $\{c,d\} : \{\delta(c,0),\delta(d,0)\} = \{b,e\}$ is unmarked, so also check 1.

6. $\{c,d\} : \{\delta(c,1),\delta(d,1)\} = \{f,a\}$ is unmarked, so do not mark.

However, in this iteration nothing new was marked, meaning that there is no point in continuing with any further iteration, and we terminate.

The Resulting Minimal Quotient Automaton

From Figure 13 only the pairs $\{a,f\}$, $\{b,e\}$ and $\{c,d\}$ are unmarked. This means that $a \approx f$, $b \approx e$ and $c \approx d$, and from following the definition of a quotient automaton we obtain the automaton in Figure 14.

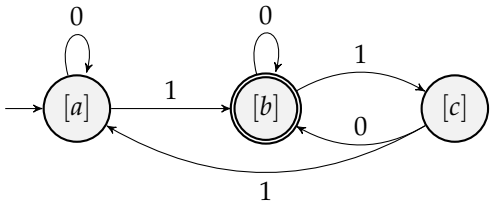

Figure 14: The minimal quotient automaton $M_{/\approx}$.