COMP30026 Models of Computation

Regular Languages

Harald Søndergaard

Lecture 15

Semester 2, 2018

DFAs vs NFAs

Surprisingly, for finite automata, adding the non-determinism does not result in more computing power.

The class of languages recognised by NFAs is exactly the class of regular languages.

Theorem: Every NFA has an equivalent DFA.

The proof rests on the so-called subset construction.

Given NFA N, we construct DFA M, each of whose states is a set of N-states.

If N has k states then M may have up to 2^k states (but it will often have far fewer than that).

DFAs vs NFAs

Consider the NFA on the right. We can systematically construct an equivalent DFA from the NFA.

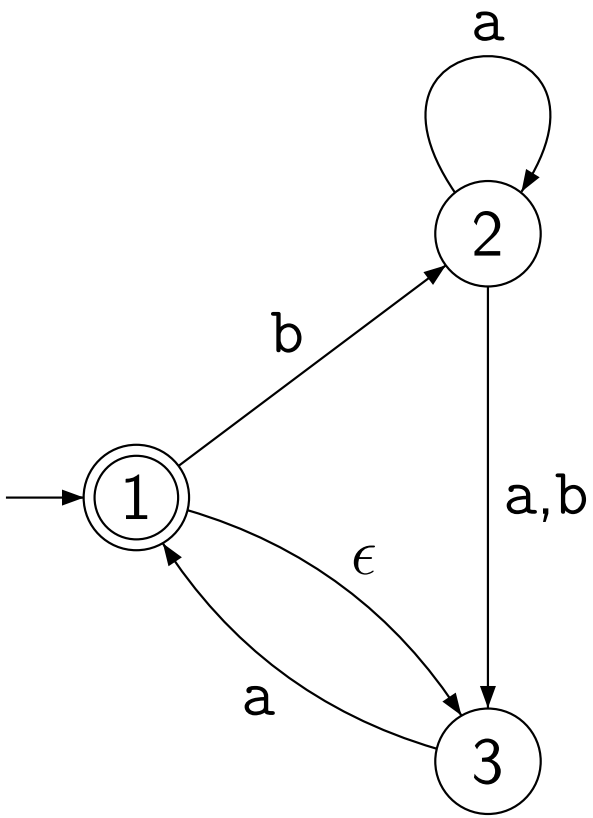
The DFA's start state is $\{1, 3\}$.

From this state, a will take us back to $\{1, 3\}$.

From $\{1, 3\}$, b can only take us to $\{2\}$.

Continuing thus (in a demand-driven fashion) gives the DFA.

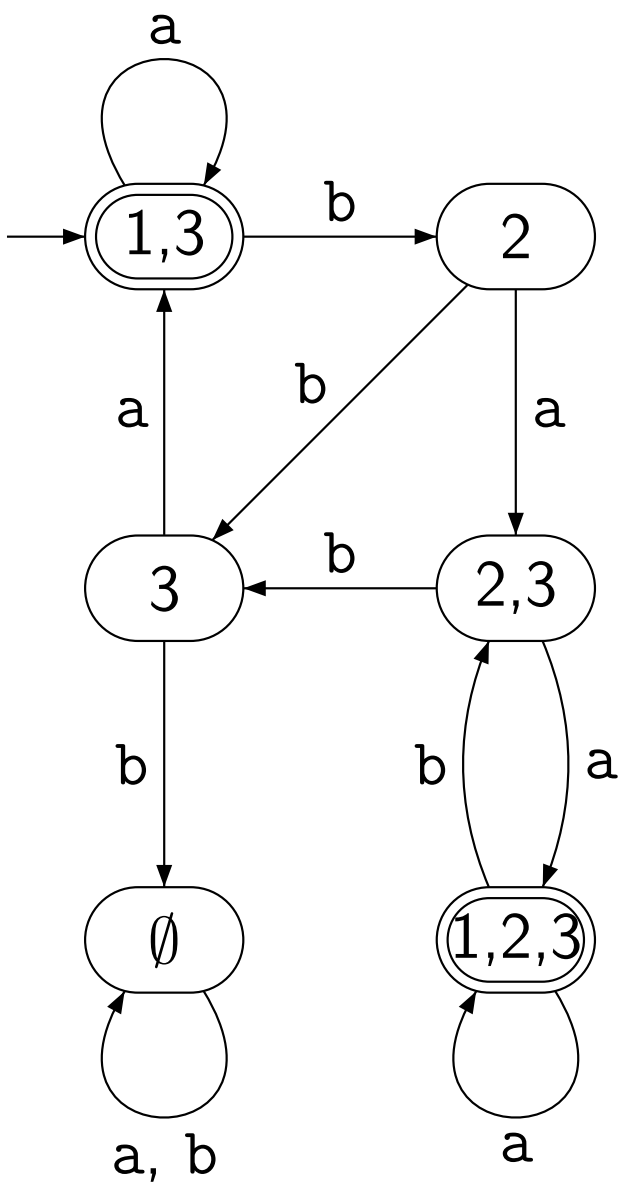
Any state S which contains an accept state from the NFA will be an accept state for the DFA.



DFAs vs NFAs

Here is the equivalent DFA that we derive.

Any state S which contains an accept state from the NFA (in this case the NFA has just one, namely state 1) becomes an accept state for the DFA.



More Formally . . .

Let $N = (Q, \Sigma, \delta, q_0, F)$. Let $\stackrel{*}{\rightarrow}_\epsilon$ be the reflexive transitive closure of $\rightarrow_\epsilon$, which in turn is defined by $s \rightarrow_\epsilon s'$ iff $s' \in \delta(s, \epsilon)$.

Let $E(S)$ be the "ϵ closure" of $S \subseteq Q$, that is, S together with all states reachable from states in S, using only ϵ steps:

$$E(S) = \bigcup_{s \in S} \{s' \in Q \mid s \stackrel{*}{\rightarrow}_\epsilon s'\}$$

We construct $M = (\mathcal{P}(Q), \Sigma, \delta', q_0', F')$ as follows:

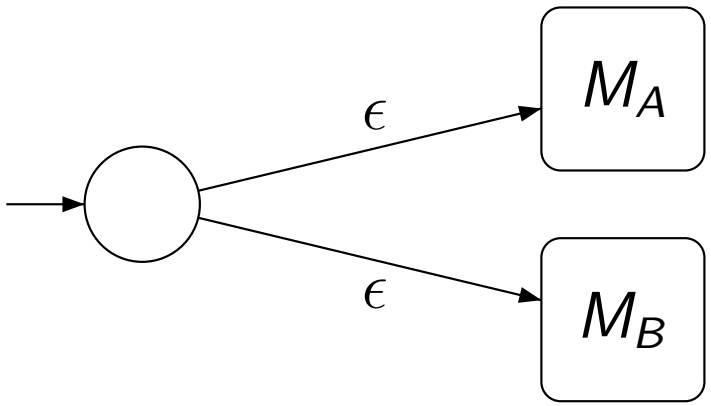
- $q_0' = E(\{q_0\})$.
- $F' = \{S \subseteq Q \mid S \cap F \neq \emptyset\}$.
- $\delta'(S, v) = \bigcup_{s \in S} E(\delta(s, v))$.

Note: This construction may include unreachable states.

Closure Results for Regular Languages

Theorem: The class of regular languages is closed under union.

Proof: Let A and B be regular languages, with recognisers M_A and M_B. An NFA that recognises $A \cup B$ is easily constructed:

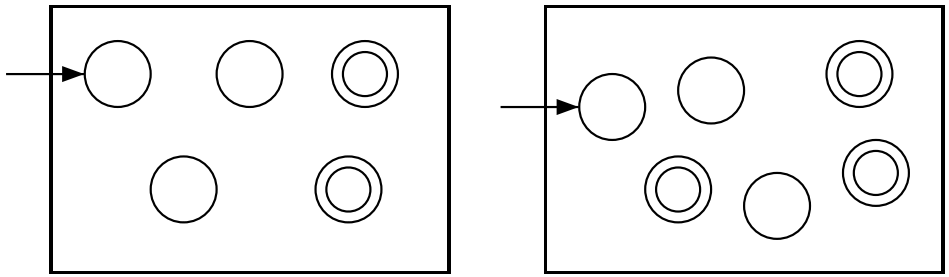


The ϵ-transitions go to the start states of M_A and M_B.

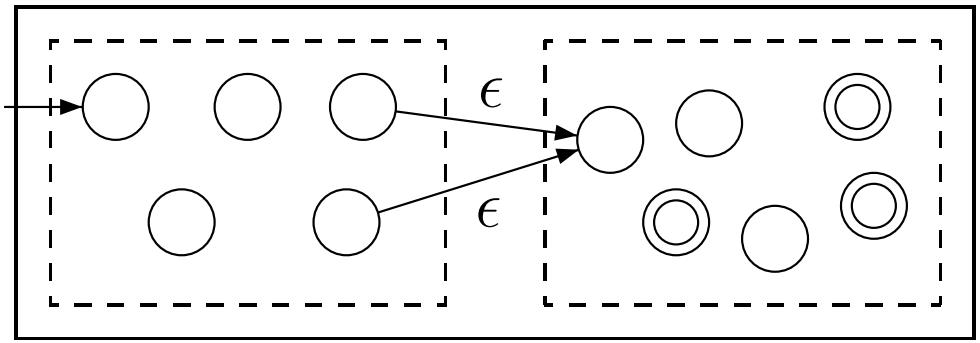
Closure Results for Regular Languages

Theorem: The class of regular languages is closed under $\circ$.

Proof: Let A and B be regular languages with these recognisers:



From these we can easily construct an NFA that recognises $A \circ B$:



The Last Construction, Formally

Let recognisers for A and B be these DFAs, respectively:

- $M_A = (Q, \Sigma, \delta, q_0, F)$
- $M_B = (Q', \Sigma, \delta', q_0', F')$

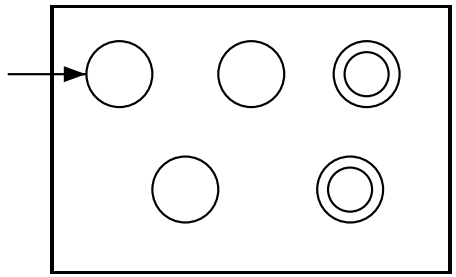
Then a recogniser for $A \circ B$ is the NFA $(Q \cup Q', \Sigma, \delta'', q_0, F')$, where

$$\delta''(q, v) = \begin{cases} \{\delta'(q, v)\} & \text{if } q \in Q' \text{ and } v \in \Sigma \\ \{\delta(q, v)\} & \text{if } q \in Q \text{ and } v \in \Sigma \\ \{q_0'\} & \text{if } q \in F \text{ and } v = \epsilon \end{cases}$$

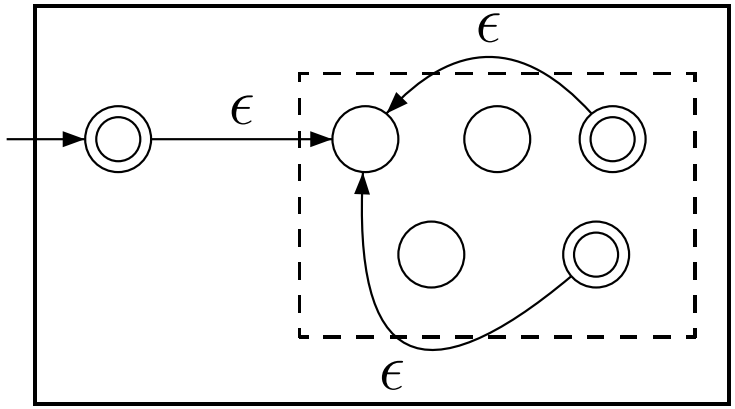
Closure Results for Regular Languages

Theorem: The class of regular languages is closed under Kleene star.

Proof: Let A be a regular language with the recogniser shown on the right.



Here is how we construct an NFA to recognise A^*:



Closure Results for Regular Languages

Regular languages have several other closure properties.

They are closed under

- intersection,
- complement, A^c
- difference (this follows, as $A \setminus B = A \cap B^c$)
- reversal.

Algorithms for Manipulating Automata

For some of these closure results, we will use the tutorials to develop useful DFA manipulation algorithms.

For this reason, the Week 8 exercises are very important.

You will see, for example, how to systematically build DFAs for languages $A \cap B$, out of DFAs for A and B.

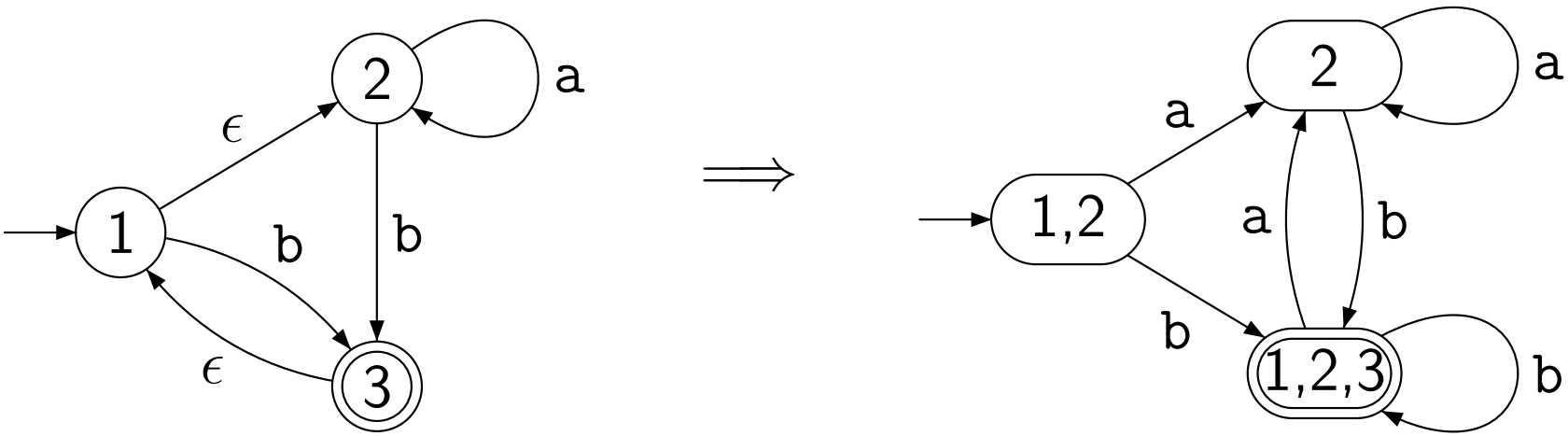
Equivalence of DFAs

We can always find a minimal DFA for a given regular language (by minimal we mean having the smallest possible number of states).

Since a DFA has a unique start state and the transition function is total and deterministic, we can test two DFAs for equivalence (modulo the names used for their states) by minimizing them.

Minimizing DFAs

There is no guarantee that DFAs that are produced by the various algorithms, such as the subset construction method, will be minimal.



Generating a Minimal DFA

The following algorithm takes an NFA and produces an equivalent minimal DFA. Of course the input can also be a DFA.

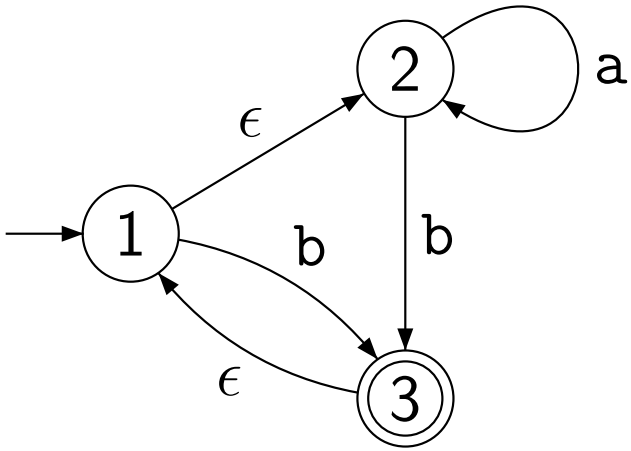
1. Reverse the NFA;
2. Determinize the result;
3. Reverse again;
4. Determinize.

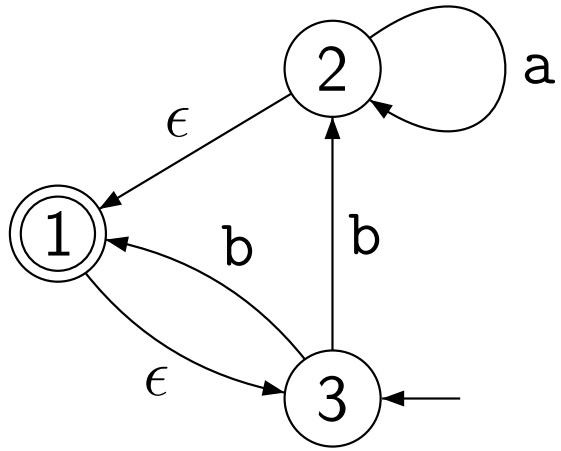
To reverse an NFA A with initial state q_0 and accept states $F \neq \emptyset$:

1. If $F = \{q\}$, let q be the accept state.
2. Otherwise add a new state q which becomes the only accept state; then, for each state in F, add an ϵ transition to q.
3. Reverse every transition in the resulting NFA, making q the initial state and q_0 the (only) accept state.

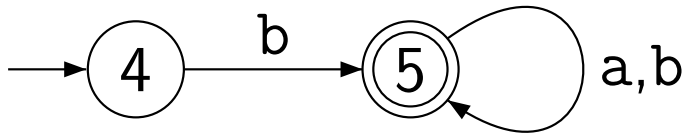
Minimization Example

Consider the NFA that we determinized two slides ago. Here it is on the left, with its reversal on the right:





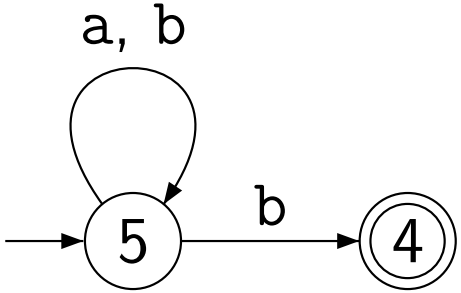
Making the reversed NFA deterministic:



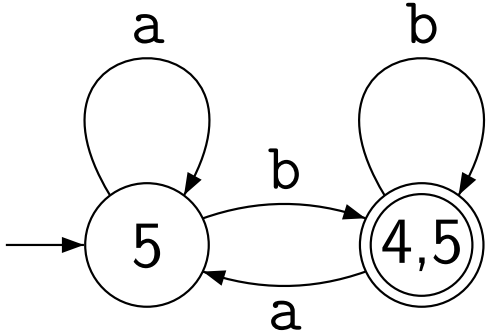
We renamed the states to avoid later confusion;
4 corresponds to $\{3\}$ and 5 to $\{1, 2, 3\}$.

Minimization Example

Now reversing the result:



And finally making the result deterministic:



Coming to a Theatre Near You

We next look at regular expressions—another way to capture the regular languages.

We shall also introduce a technique called "pumping", for proving that a language is not regular.