Equivalence of DFAs and NFAs

NFA are easier to build than DFA because one does not have to worry, for any state, of having out-going edges carrying a unique label.

The surprising thing is that NFA and DFA actually have the same expressiveness, i.e. all that can be defined by means of a NFA can also be defined with a DFA (the converse is trivial since a DFA is already a NFA).

More precisely, there is a procedure, called **the subset construction**, which converts any NFA to a DFA.

Subset construction

Consider that, in a NFA, from a state q with several out-going edges with the same label a, the transition function δ_N leads, in general, to *several* states.

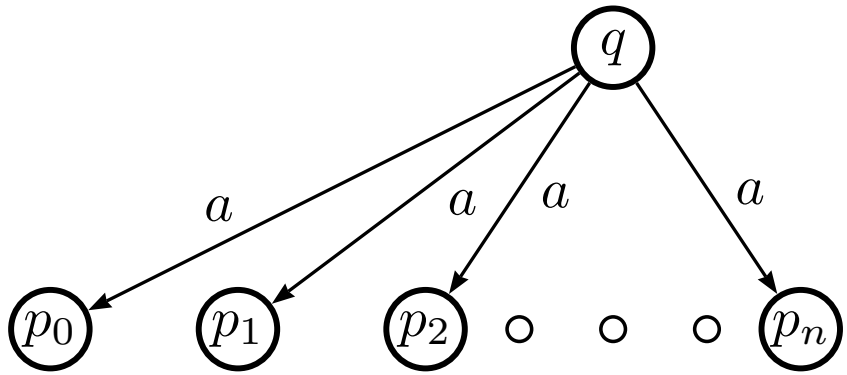
The idea of the subset construction is to create a new automaton where these edges are merged.

So we create a state p which corresponds to the set of states $\delta_N(q, a)$ in the NFA. Accordingly, we create a state r which corresponds to the set $\{q\}$ in the NFA. We create an edge labeled a between r and p. The important point is that *this edge is unique*.

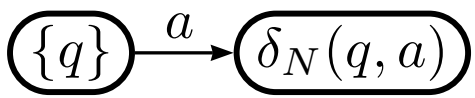
This is the first step to create a DFA from a NFA.

Subset construction (cont)

Graphically, instead of the non-determinism



where $\delta_N(q, a) = \{p_0, p_1, \ldots, p_n\}$, we get the determinism



Subset construction (cont)

Now, let us present the complete algorithm for the subset construction.

Let us start from a NFA $\mathcal{N} = (Q_N, \Sigma, \delta_N, q_0, F_N)$.

The goal is to construct a DFA $\mathcal{D} = (Q_D, \Sigma, \delta_D, \{q_0\}, F_D)$ such that $L(\mathcal{D}) = L(\mathcal{N})$.

Notice that the input alphabet of the two automata are the same and the initial state of $\mathcal{D}$ if the set containing only the initial state of $\mathcal{N}$.

The other components of $\mathcal{D}$ are constructed as follows.

- Q_D is the set of subsets of Q_N; i.e. Q_D is the **power set** of Q_N. Thus, if Q_D has n states, Q_D has 2^n states. Fortunately, often not all these states are **accessible** from the initial state of Q_D, so these inaccessible states can be discarded.

Subset construction (cont)

Why is 2^n the number of subsets of a finite set of cardinal n?

Let us order the n elements. Let us represent each subset by an n-bit string where bit i corresponds to the i-th element: it is 1 if the i-th element is present in the subset and 0 if not.

This way, we counted all the subsets and not more (a bit cannot always be 0 since all elements are used to form subsets and cannot always be 1 if there is more than one element).

There are 2 possibilities, 0 or 1, for the first bit; 2 possibilities for the second bit etc. Since the choices are independent, we multiply all of them: $\underbrace{2 \times 2 \times \cdots \times 2}_{n \text{ times}} = 2^n$.

Hence the number of subsets of an n-element set is also 2^n.

Subset construction (cont)

Resuming the definition of DFA $\mathcal{D}$, the other components are defined as follows.

- F_D is the set of subsets S of Q_N such that $S \cap F_N \neq \emptyset$. That is, F_D is all sets of N's states that include at least one final state of $\mathcal{N}$.
- for each set $S \subseteq Q_N$ and for each input $a \in \Sigma$,

$$\delta_D(S, a) = \bigcup_{q \in S} \delta_N(q, a)$$

  In other words, to compute $\delta_D(S, a)$ we look at all the states q in S, see what states of $\mathcal{N}$ are reached from q on input a and take the union of all those states to make the next state of $\mathcal{D}$.

Subset construction/Example/Transition table

Let us consider the NFA given by its transition table page 106:

NFA $\mathcal{N}$	0	1
$\to q_0$	$\{q_0, q_1\}$	$\{q_0\}$
q_1	$\varnothing$	$\{q_2\}$
$\#q_2$	$\varnothing$	$\varnothing$

and let us create an equivalent DFA.

First, we form all the subsets of the sets of the NFA and put them in the first column:

DFA $\mathcal{D}$	0	1
$\varnothing$		
$\{q_0\}$		
$\{q_1\}$		
$\{q_2\}$		
$\{q_0, q_1\}$		
$\{q_0, q_2\}$		
$\{q_1, q_2\}$		
$\{q_0, q_1, q_2\}$		

Subset construction/Example/Transition table (cont)

Then we annotate in this first column the states with $\rightarrow$ if and only if they contain the initial state of the NFA, here q_0, and we add a $\#$ if and only if the states contain at least a final state of the NFA, here q_2.

DFA $\mathcal{D}$	0	1
$\varnothing$		
$\rightarrow\{q_0\}$		
$\{q_1\}$		
$\#\{q_2\}$		
$\{q_0, q_1\}$		
$\#\{q_0, q_2\}$		
$\#\{q_1, q_2\}$		
$\#\{q_0, q_1, q_2\}$		

Subset construction/Example/Transition table (cont)

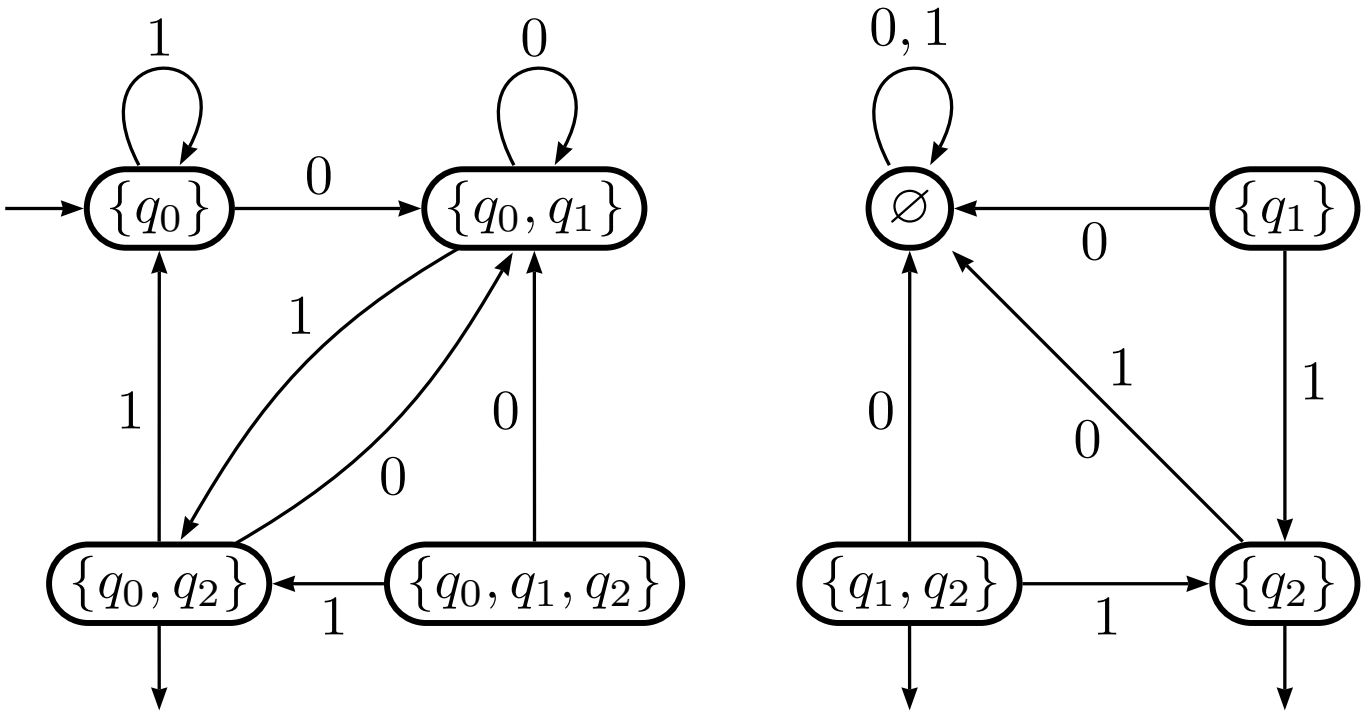
DFA $\mathcal{D}$	0	1
$\varnothing$	$\varnothing$	$\varnothing$
$\rightarrow\{q_0\}$	$\delta_N(q_0, 0)$	$\delta_N(q_0, 1)$
$\{q_1\}$	$\delta_N(q_1, 0)$	$\delta_N(q_1, 1)$
$\#\{q_2\}$	$\delta_N(q_2, 0)$	$\delta_N(q_2, 1)$
$\{q_0, q_1\}$	$\delta_N(q_0, 0) \cup \delta_N(q_1, 0)$	$\delta_N(q_0, 1) \cup \delta_N(q_1, 1)$
$\#\{q_0, q_2\}$	$\delta_N(q_0, 0) \cup \delta_N(q_2, 0)$	$\delta_N(q_0, 1) \cup \delta_N(q_2, 1)$
$\#\{q_1, q_2\}$	$\delta_N(q_1, 0) \cup \delta_N(q_2, 0)$	$\delta_N(q_1, 1) \cup \delta_N(q_2, 1)$
$\#\{q_0, q_1, q_2\}$	$\delta_N(q_0, 0) \cup \delta_N(q_1, 0) \cup \delta_N(q_2, 0)$	$\delta_N(q_0, 1) \cup \delta_N(q_1, 1) \cup \delta_N(q_2, 1)$

Subset construction/Example/Transition table (cont)

DFA $\mathcal{D}$	0	1
$\varnothing$	$\varnothing$	$\varnothing$
$\rightarrow\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_1\}$	$\varnothing$	$\{q_2\}$
$\#\{q_2\}$	$\varnothing$	$\varnothing$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\#\{q_0, q_2\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\#\{q_1, q_2\}$	$\varnothing$	$\{q_2\}$
$\#\{q_0, q_1, q_2\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$

Subset construction/Example/Transition diagram

The transition diagram of the DFA $\mathcal{D}$ is then

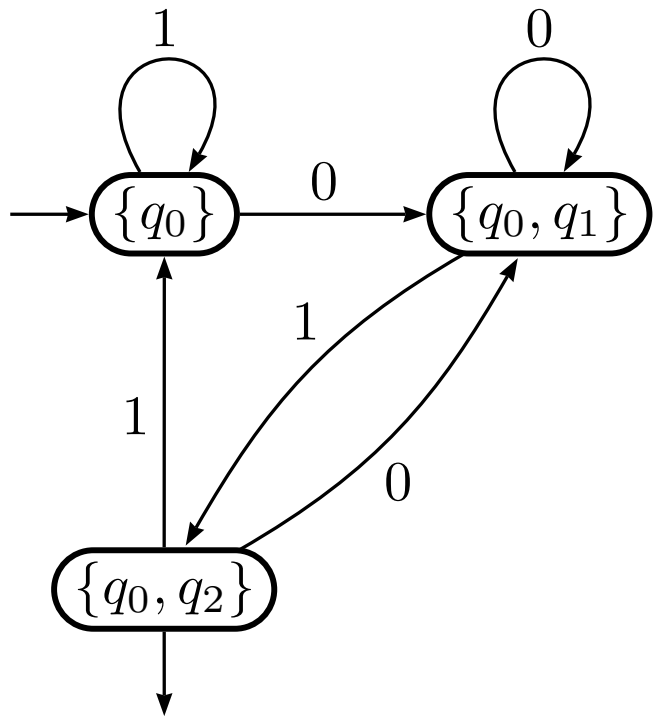


where states with out-going edges which have no end are final states.

Subset construction/Example/Transition diagram (cont.)

If we look carefully at the transition diagram, we see that the DFA is actually made of two parts which are disconnected, i.e. not joined by an edge.

In particular, since we have only one initial state, this means that one part is not accessible, i.e. some states are never used to recognise or reject an input word, and we can remove this part.



Subset construction/Example/Transition diagram (cont.)

It is important to understand that the states of the DFA are subsets of the NFA states.

This is due to the construction and, when finished, it is possible to hide this by **renaming the states**. For example, we can rename the states of the previous DFA in the following manner: $\{q_0\}$ into A, $\{q_0, q_1\}$ in B and $\{q_0, q_2\}$ in C.

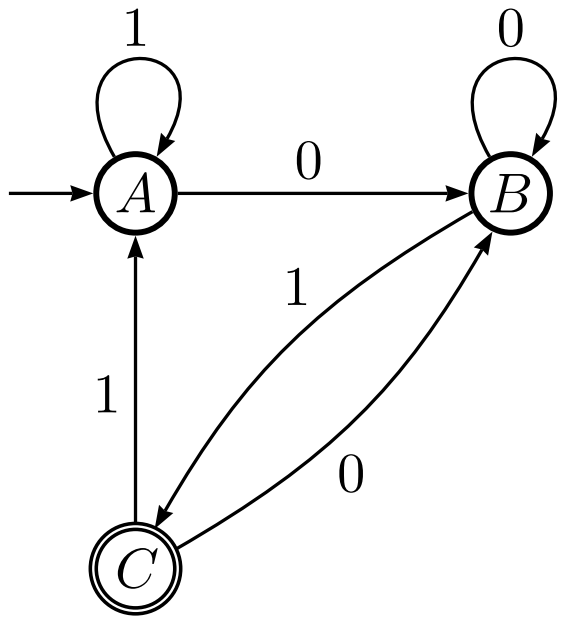
So the transition table changes:

DFA $\mathcal{D}$	0	1
$\rightarrow\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\#\{q_0, q_2\}$	$\{q_0, q_1\}$	$\{q_0\}$

DFA $\mathcal{D}$	0	1
$\rightarrow A$	B	A
B	B	C
$\#C$	B	A

Subset construction/Example/Transition diagram (cont.)

So, finally, the DFA is simply



Subset construction/Optimisation

Even if in the worst case the resulting DFA has an exponential number of states of the corresponding NFA, it is in practice often possible to avoid the construction of inaccessible states.

- The singleton containing the initial state (in our example, $\{q_0\}$) is accessible.
- Assume we have a set S of accessible states; then for each input symbol a, we compute $\delta_D(S, a)$: this new set is also accessible.
- Repeat the last step, starting with $\{q_0\}$, until no new (accessible) sets are found.

Subset construction/Optimisation/Example

Let us consider the NFA given by its transition table page 106:

NFA $\mathcal{N}$	0	1
$\rightarrow q_0$	$\{q_0, q_1\}$	$\{q_0\}$
q_1	$\varnothing$	$\{q_2\}$
$\#q_2$	$\varnothing$	$\varnothing$

Initially, the sole subset of accessible states is $\{q_0\}$:

DFA $\mathcal{D}$	0	1
$\rightarrow\{q_0\}$	$\delta_N(q_0, 0)$	$\delta_N(q_0, 1)$

that is

DFA $\mathcal{D}$	0	1
$\rightarrow\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$

Subset construction/Optimisation/Example (cont)

Therefore $\{q_0, q_1\}$ and $\{q_0\}$ are accessible sets. But $\{q_0\}$ is not a new set, so we only add to the table entries $\{q_0, q_1\}$ and compute the transitions from it:

DFA $\mathcal{D}$	0	1
$\rightarrow\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$

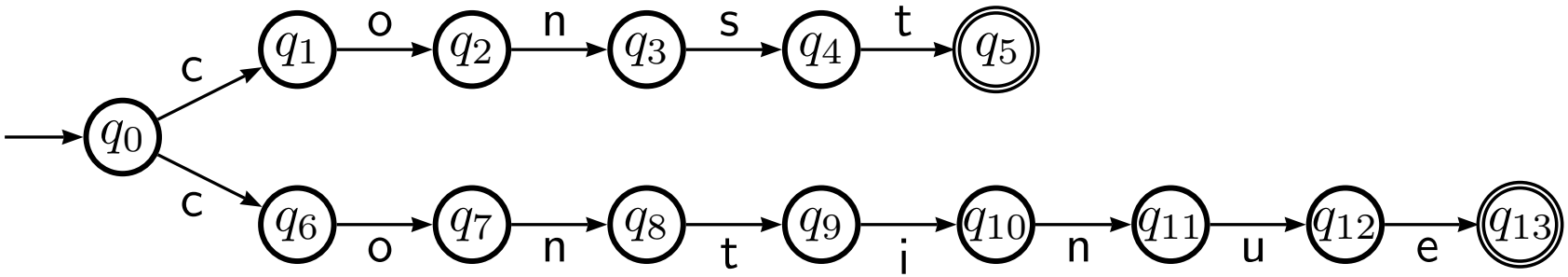
This step uncovered a new set of accessible states, $\{q_0, q_2\}$, which we add to the table and repeat the procedure, and mark it as final state since $q_2 \in \{q_0, q_2\}$:

DFA $\mathcal{D}$	0	1
$\rightarrow\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\#\{q_0, q_2\}$	$\{q_0, q_1\}$	$\{q_0\}$

We are done since there is no more new accessible sets.

Subset construction/Tries

Compilers try to recognise a prefix of the input character stream (i.e the first meaningful unit, called **lexeme**, of the given program). Consider the C keywords **const** and **continue**:

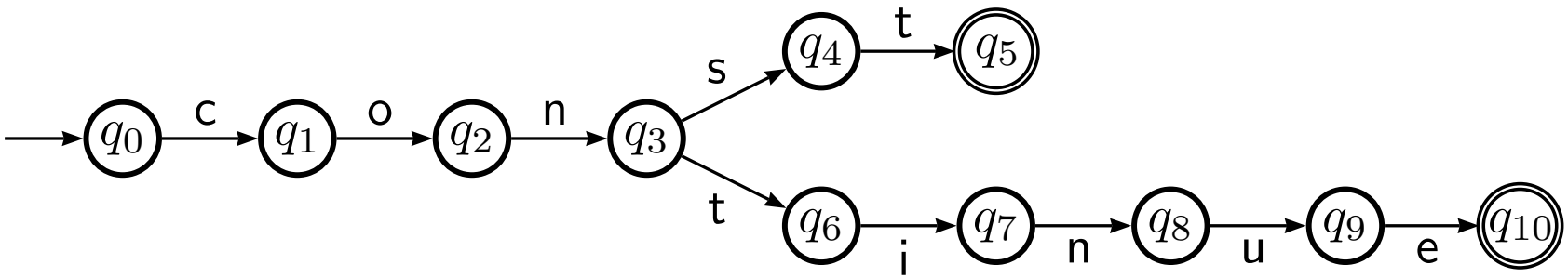


This example shows that a NFA is much more comfortable than a DFA for specifying lexemes: we design *separately* the automata for each kind of lexeme, called **token**, and then merge their initial states into one, leading to one (possibly big) NFA.

It is possible to apply the subset construction to this NFA.

Subset construction/Tries (cont)

After forming the corresponding NFA as in the previous example, it is actually easy to construct an equivalent DFA by **sharing their prefixes**, hence obtaining a tree-like automaton called **trie** (pronounced as the word ‘try’):



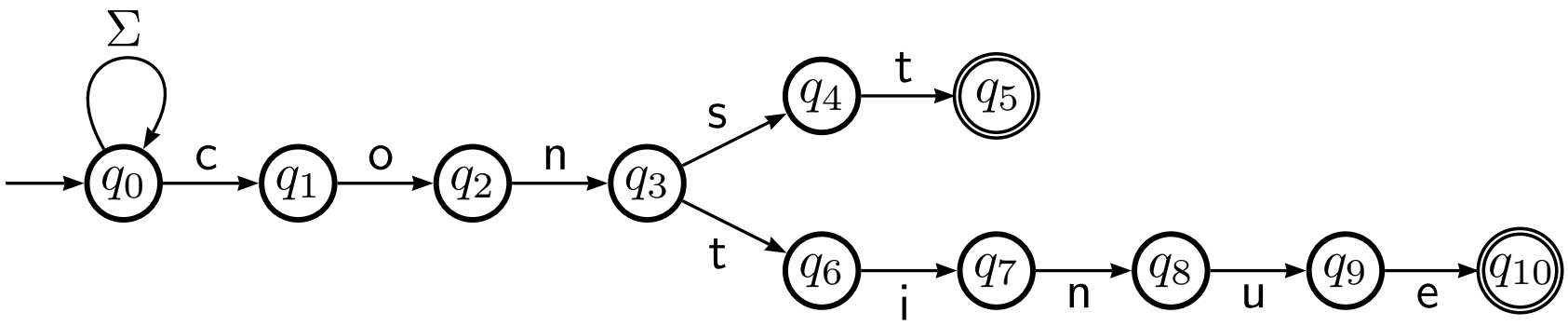
The origin of this word comes from the word re**trie**val.

Note that this construction only works for a list of constant words, like keywords.

Subset construction/Text searching

This technique can easily be generalized for searching constant strings (like keywords) in a text, i.e. not only as a prefix of a text, but *at any position*.

It suffices to add a loop on the initial state for each possible input symbol. If we note Σ the language alphabet, we get the NFA



Subset construction/Text searching (cont)

It is possible to apply the subset construction to this NFA or to use it directly for searching *all occurrences* of the two keywords at *any position* in a text.

In case of direct use, the difference between this NFA and the trie page 128 is that there is no need here to “restart” by hand the recognition process once a keyword has been recognised: we just do not stop on final states but continue reading the input after discarding the recognised prefix (at the last final state).

Subset construction/Text searching (cont)

This works because of the loop on the initial state.

Try for instance the input `constantcontinue`. We have $\hat{\delta}(q_0, \texttt{const}) = \{q_0, q_5\}$. Since q_5 is a final state, we recognised the input `const`.

But we can continue because $\hat{\delta}(q_0, \texttt{consta}) = \{q_0\}$ etc., until we reach the end of the input.

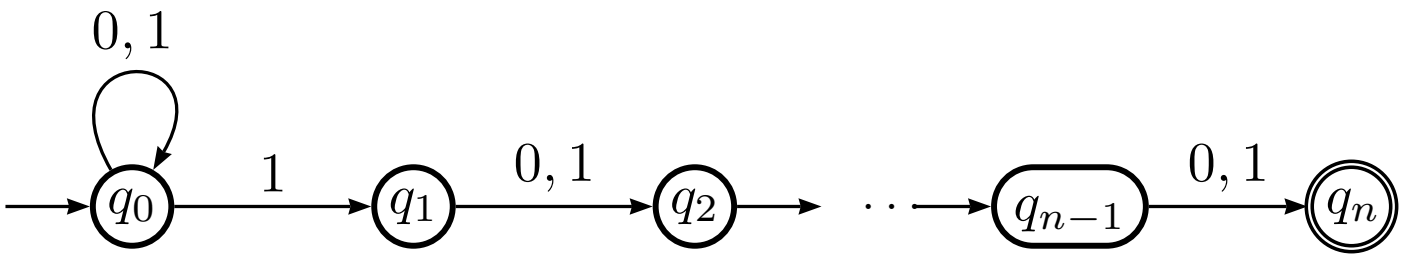
Subset construction/Bad case

The subset construction can lead, in the worst case, to a number of states which is the total number of state subsets of the NFA.

In other words, if the NFA has n states, the equivalent DFA by subset construction can have 2^n states (see page 114 for the count of all the subsets of a finite set).

Subset construction/Bad case (cont)

Consider the following NFA, which recognises all binary strings which have 1 at the *n*-th position from the end:



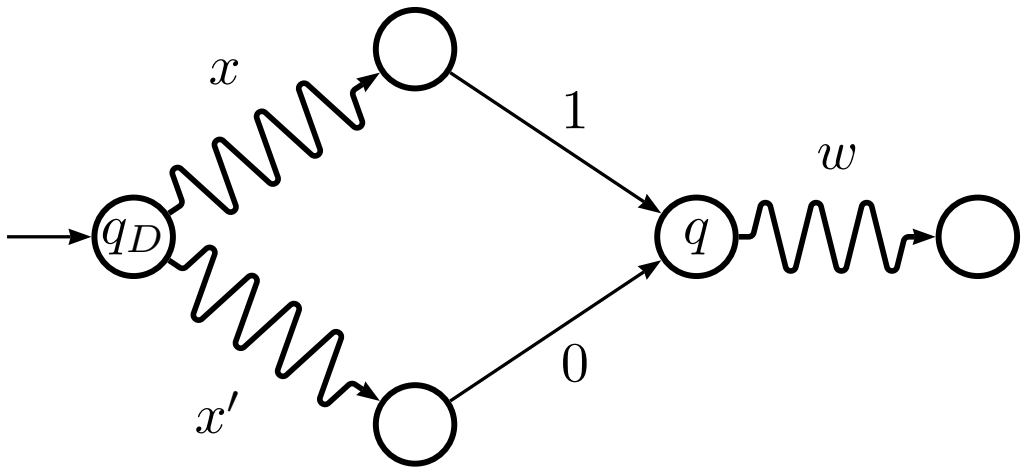
The language recognised by this NFA is $\Sigma^* 1 \Sigma^{n-1}$, where $\Sigma = \{0, 1\}$, that is: all words of length greater or equal to *n* are accepted as long as the *n*-th bit from the **end** is 1.

Therefore, in any equivalent DFA, all the prefixes of length *n* should not lead to a stuck state, because the automaton must wait until the **end** of the word to accept or reject it.

Subset construction/Bad case (cont)

If the states reached by these prefixes are all different, then there are at least 2^n states in the DFA.

Equivalently (by contraposition), if there are less than 2^n states, then some states can be reached by several strings of length n:



where words $x1w$ and $x'0w$ have length n.

Subset construction/Bad case (cont)

Let us call the DFA $\mathcal{D} = (Q_D, \Sigma, \delta_D, q_D, F_D)$, where $q_D = \{q_0\}$.

The extended transition function is noted $\hat{\delta}_D$ as usual. The situation of the previous picture can be formally expressed as

$$\hat{\delta}_D(q_D, x1) = \hat{\delta}_D(q_D, x'0) = q \tag{1}$$

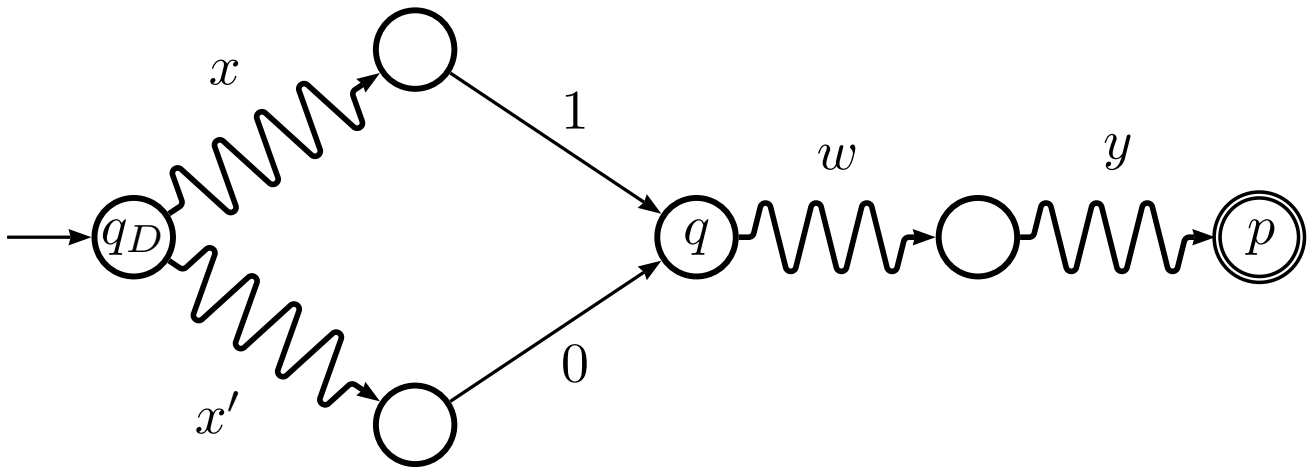
$$|x1w| = |x'0w| = n \tag{2}$$

where $|u|$ is the length of u.

Subset construction/Bad case (cont)

Let y be a any string of 0 and 1 such that $|wy| = n - 1$.

Then $\hat{\delta}_D(q_D, x1wy) \in F_D$ since there is a 1 at the n-th position from the end:



Also, $\hat{\delta}_D(q_D, x'0wy) \notin F_D$ because there is a 0 at the n-th position from the end.

Subset construction/Bad case (cont)

On the other hand, equation (1) implies

$$\hat{\delta}_D(q_D, x1wy) = \hat{\delta}_D(q_D, x'0wy) = p$$

So there is contradiction because a state (here, p) must be either final or not final, it cannot be both...

As a consequence, we must reject our initial assumption: there are at least 2^n states in the equivalent DFA.

This is a very bad case, even if it is not the worst case (2^{n+1} states).