COMP20121 The Implementation and Power of Computer Languages

‘Power’ Part

http://www.cs.man.ac.uk/~petera/2121/index.html .

Peter Aczel

room: CS2.52, tel: 56155

email: petera@cs.man.ac.uk

School of Computer Science, University of Manchester

lecture 1: Regular languages

LECTURE TWO

Finite automata for regular languages

Match, or no match?

It can be hard to work out whether a given word matches a pattern.

Match, or no match?

It can be hard to work out whether a given word matches a pattern.

Consider the pattern: $ab*(a|b)b*(a|b)*,$

Match, or no match?

It can be hard to work out whether a given word matches a pattern.

Consider the pattern: $ab*(a|b)b*(a|b)*$,

and the word: $abbababababbbababab$.

Does it match, or doesn't it?

Match, or no match?

It can be hard to work out whether a given word matches a pattern.

Consider the pattern: $ab*(a|b)b*(a|b)*$,

and the word: $abbabababbbababab$.

Does it match, or doesn't it?

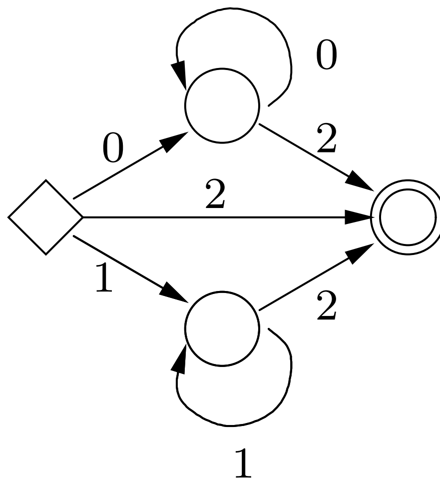
We want to match letters in the target word with letters in the pattern.

Automata

Instead of dealing with syntax, human beings often do better with pictures. e.g. when working with the pattern $(((0*)|1)2|(0|(1*))2)$: you might draw

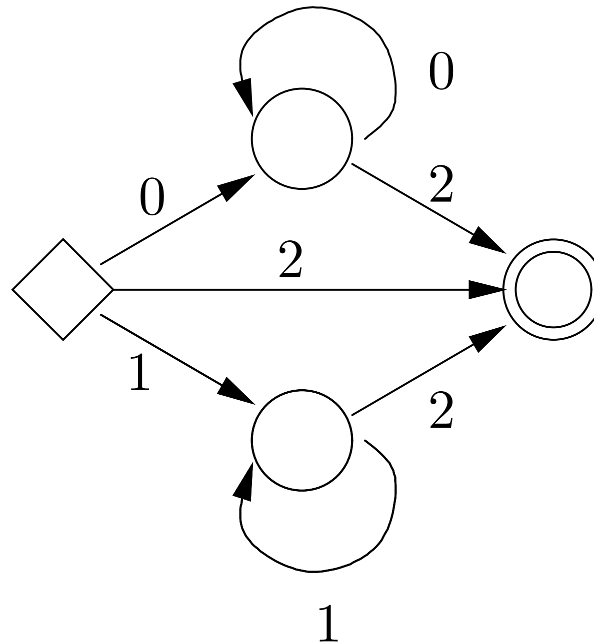
Automata

Instead of dealing with syntax, human beings often do better with pictures. e.g. when working with the pattern $(((0*)|1)2|(0|(1*))2)$: you might draw



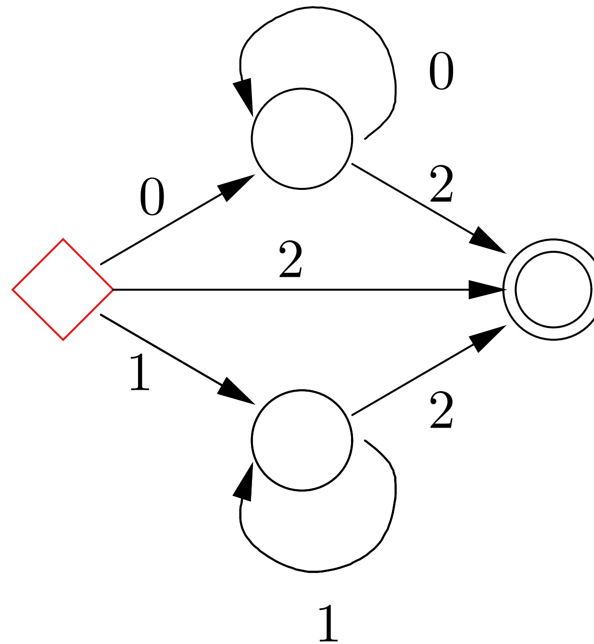
Accepting a word

We can follow any word through the automaton. Consider 0002.



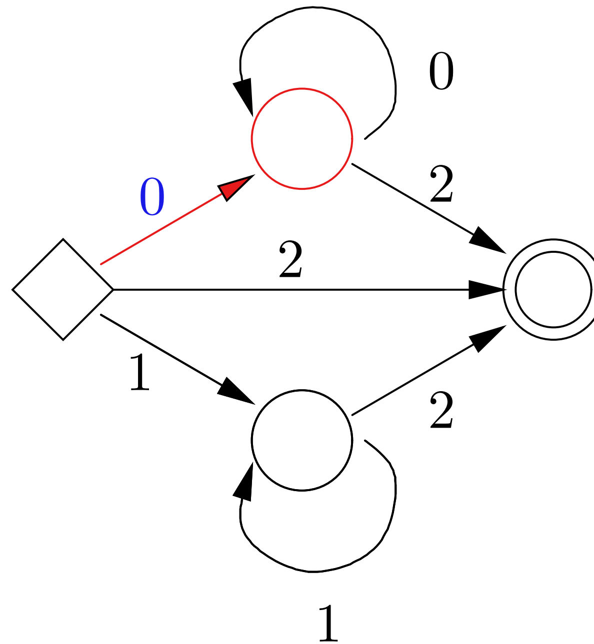
Accepting a word

We can follow any word through the automaton. Consider 0002.



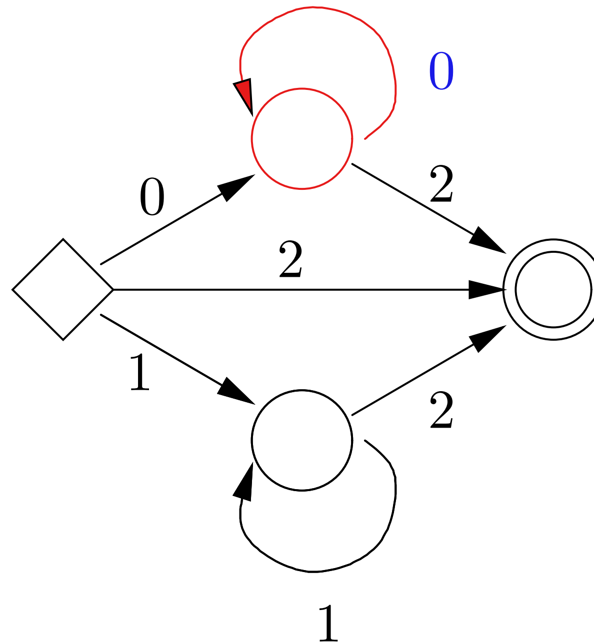
Accepting a word

We can follow any word through the automaton. Consider 0002.



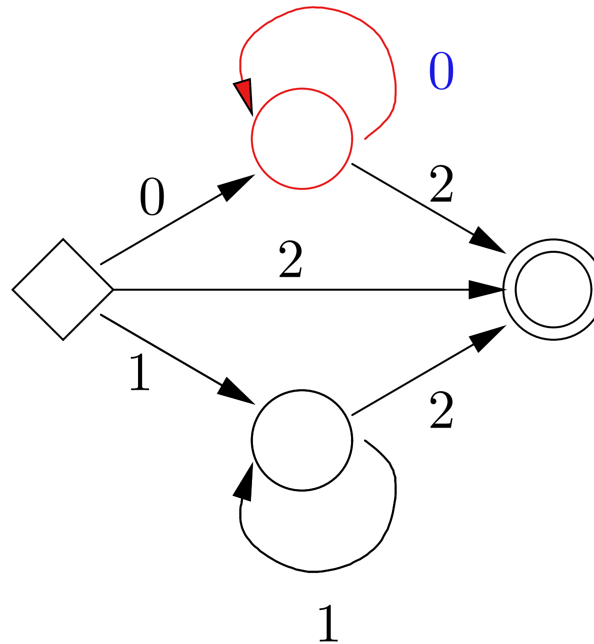
Accepting a word

We can follow any word through the automaton. Consider 0002.



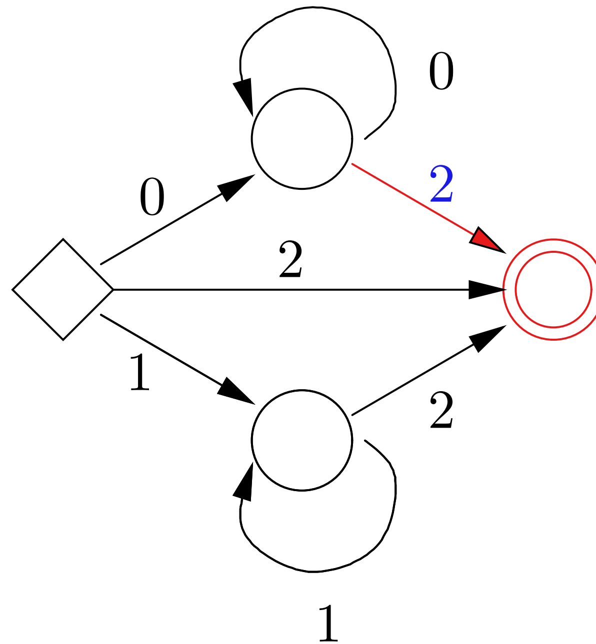
Accepting a word

We can follow any word through the automaton. Consider 0002.



Accepting a word

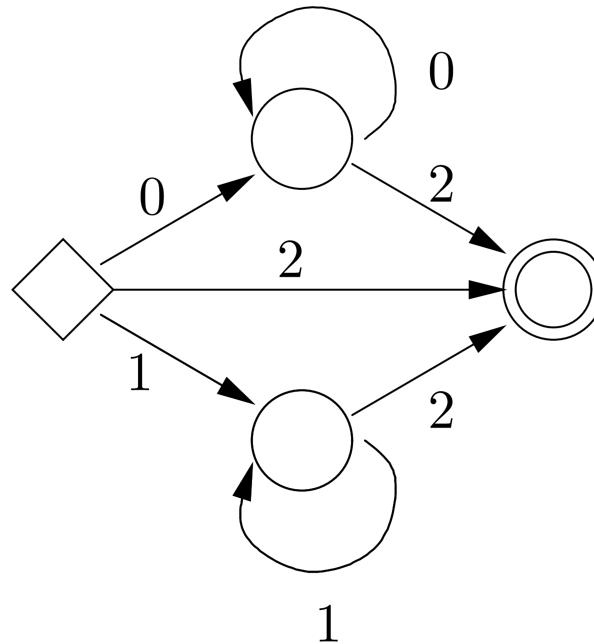
We can follow any word through the automaton. Consider 0002.



The double circle is an accepting state.

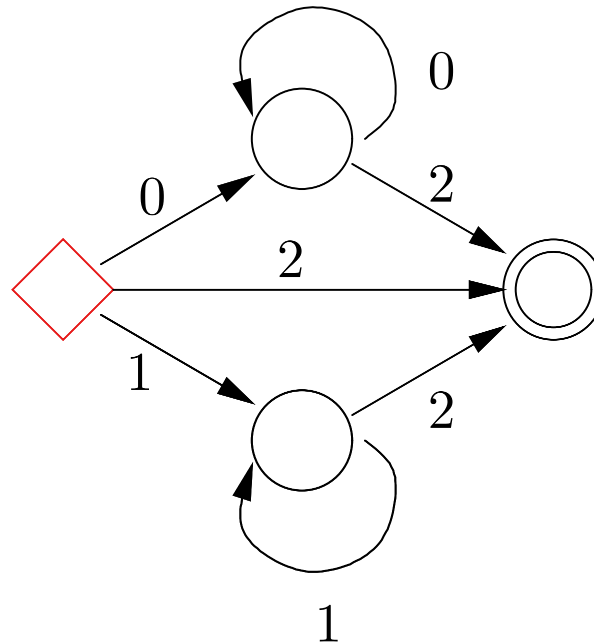
Rejecting a word-I

We can follow any word through the automaton. Consider 0001.



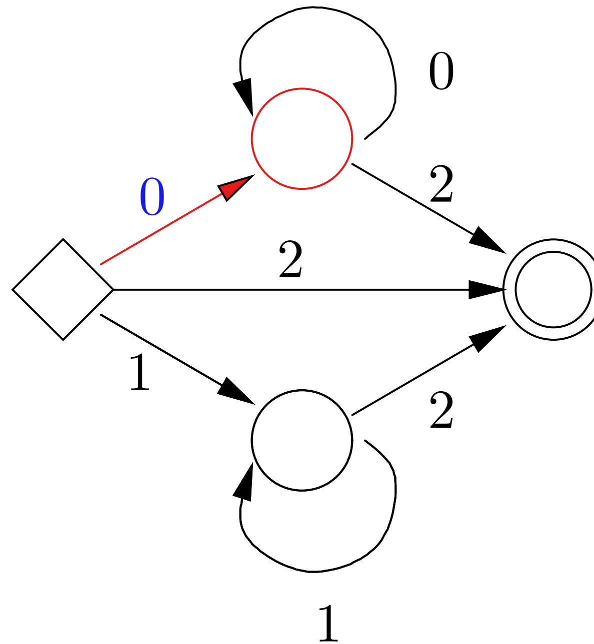
Rejecting a word-I

We can follow any word through the automaton. Consider 0001.



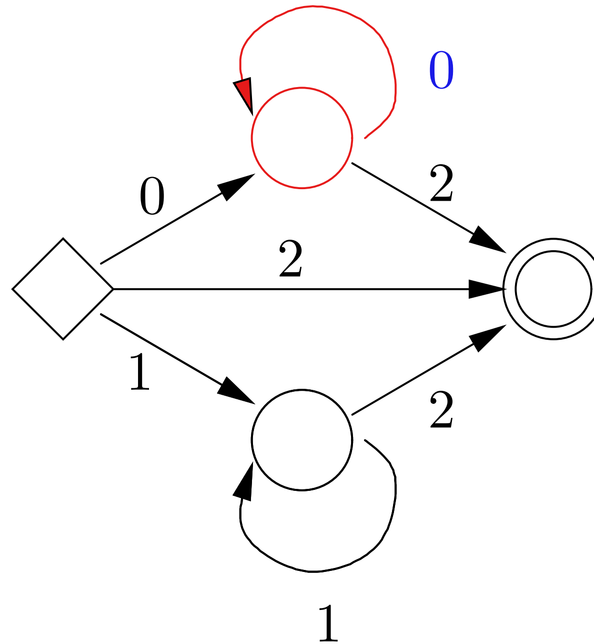
Rejecting a word-I

We can follow any word through the automaton. Consider 0001.



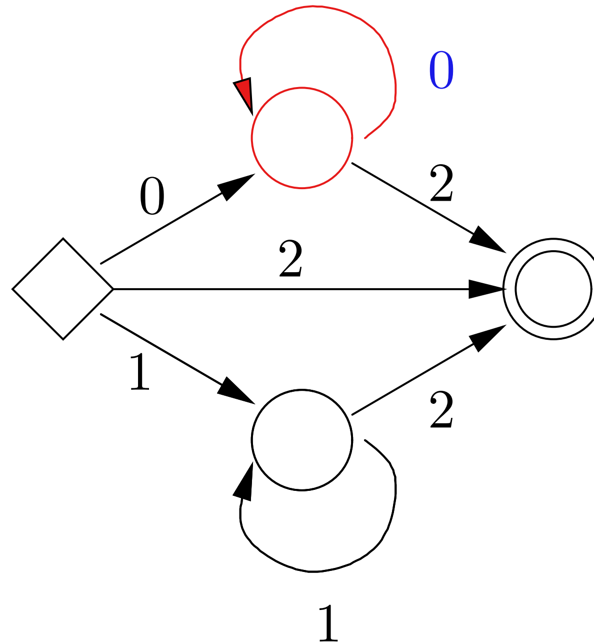
Rejecting a word-I

We can follow any word through the automaton. Consider $0\textcolor{blue}{0}01$.



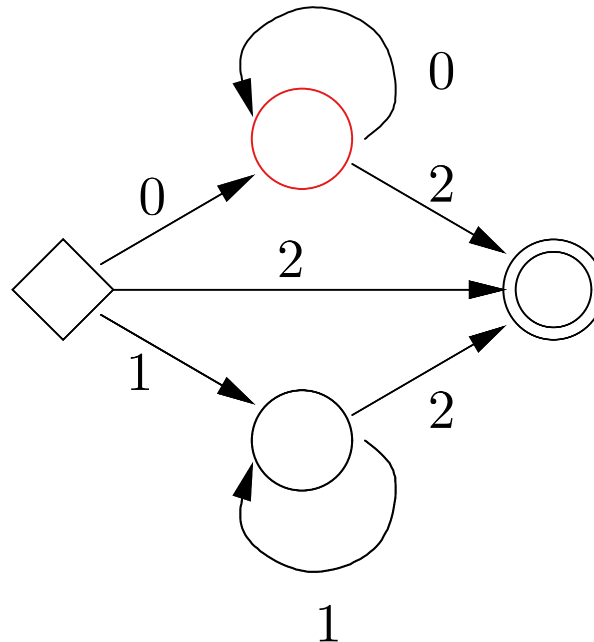
Rejecting a word-I

We can follow any word through the automaton. Consider 0001.



Rejecting a word-I

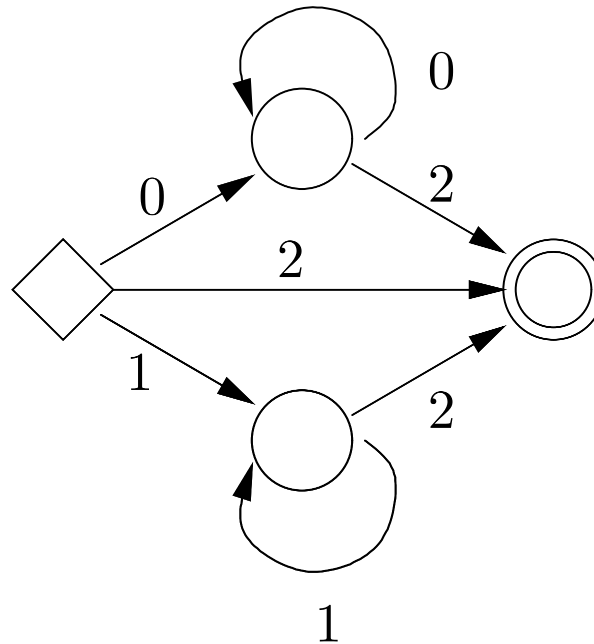
We can follow any word through the automaton. Consider 0001.



Reject this word because we are stuck.

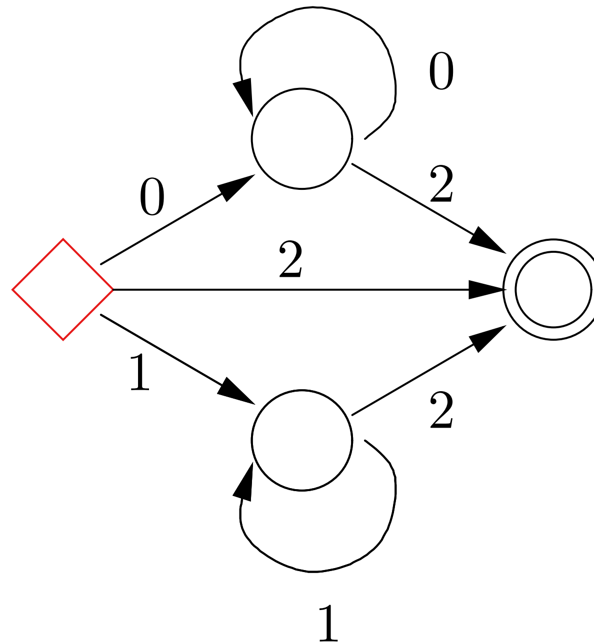
Rejecting a word-II

We can follow any word through the automaton. Consider 000.



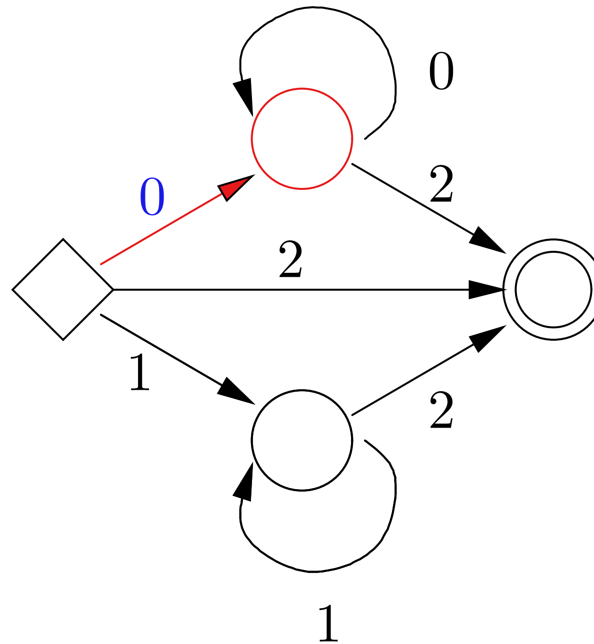
Rejecting a word-II

We can follow any word through the automaton. Consider 000.



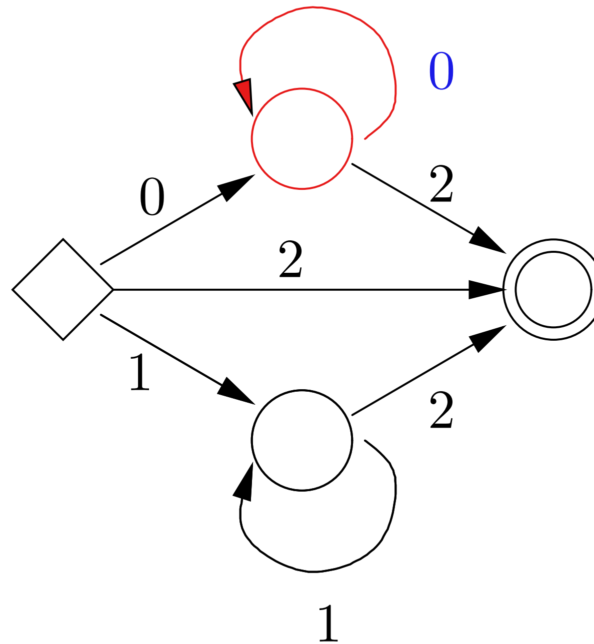
Rejecting a word-II

We can follow any word through the automaton. Consider 000.



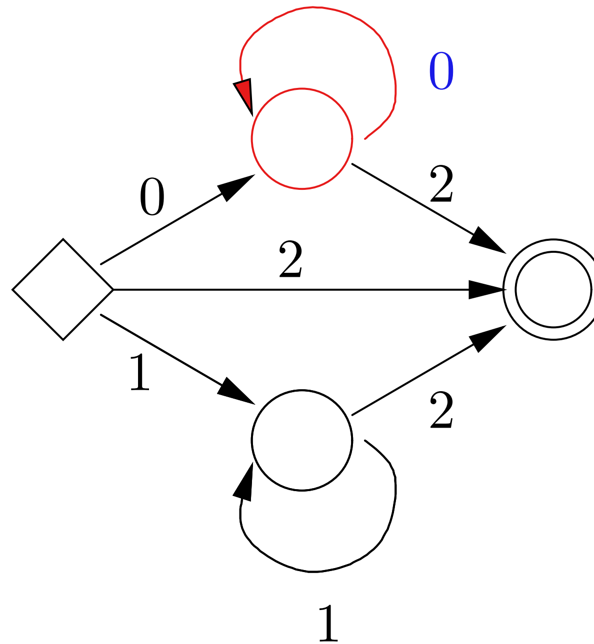
Rejecting a word-II

We can follow any word through the automaton. Consider 000.



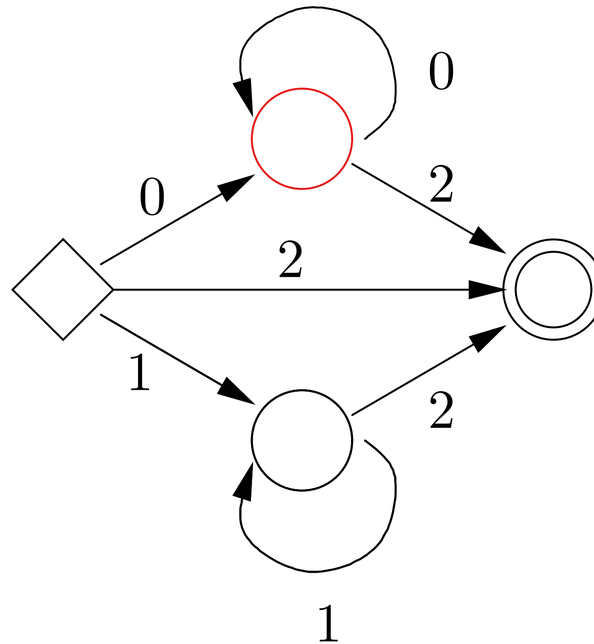
Rejecting a word-II

We can follow any word through the automaton. Consider 000.



Rejecting a word-II

We can follow any word through the automaton. Consider 000.



Reject: we end in a non-accepting state.

DFA—Definition

Definition 7 *Let* Σ *be a finite alphabet. A (deterministic) finite automaton, (DFA), over* Σ *consists of* $(Q, q_\bullet, F, \delta)$ *where:*

- Q *is a finite set of states;*

DFA—Definition

Definition 7 *Let Σ be a finite alphabet. A (deterministic) finite automaton, (DFA), over Σ consists of $(Q, q_\bullet, F, \delta)$ where:*

- Q *is a finite set of states;*
- $q_\bullet \in Q$ *is the start state;*

DFA—Definition

Definition 7 *Let Σ be a finite alphabet. A (deterministic) finite automaton, (DFA), over Σ consists of $(Q, q_\bullet, F, \delta)$ where:*

- Q *is a finite set of states;*
- $q_\bullet \in Q$ *is the start state;*
- $F \subseteq Q$ *is the set of accepting states;*

and

DFA—Definition (continued)

- δ is a transition function which for every state $q \in Q$ and every symbol $x \in \Sigma$ returns the next state $\delta(q, x) \in Q$.
- So δ is a function from $Q \times \Sigma$ to Q.
- When $\delta(q, x) = q'$ we often write

$$q \xrightarrow{x} q'.$$

NFA—Definition

Definition 8 *A non-deterministic finite automaton, (NFA), consists of* $(Q, q_\bullet, F, \delta)$ *where:*

- Q *is a a finite set of states;*

NFA—Definition

Definition 8 *A non-deterministic finite automaton, (NFA), consists of* $(Q, q_\bullet, F, \delta)$ *where:*

- Q *is a a finite set of states;*
- $q_\bullet \in Q$ *is the start state;*

NFA—Definition

Definition 8 *A non-deterministic finite automaton, (NFA), consists of* $(Q, q_\bullet, F, \delta)$ *where:*

- Q *is a a finite set of states;*
- $q_\bullet \in Q$ *is the start state;*
- $F \subseteq Q$ *is the set of accepting states;*

and

NFA—Definition (continued)

- δ is a transition relation which relates a pair consisting of a state and a letter with a state.

NFA—Definition (continued)

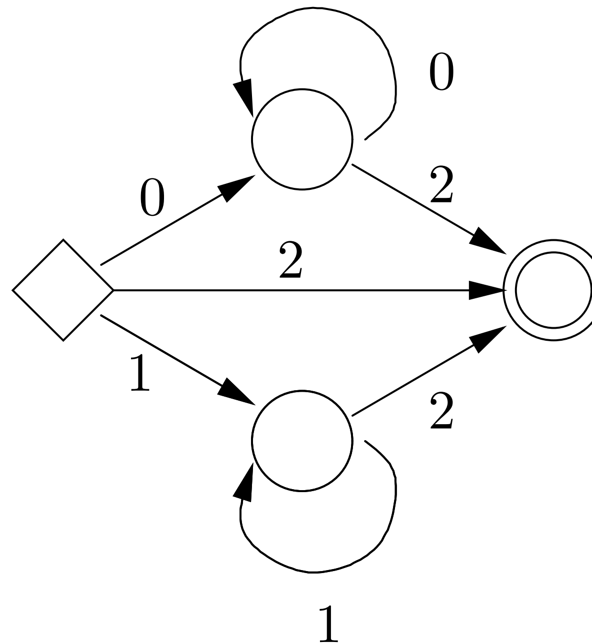
- δ is a transition relation which relates a pair consisting of a state and a letter with a state.

When (q, x) is δ-related to q' we often write

$$q \xrightarrow{x} q'.$$

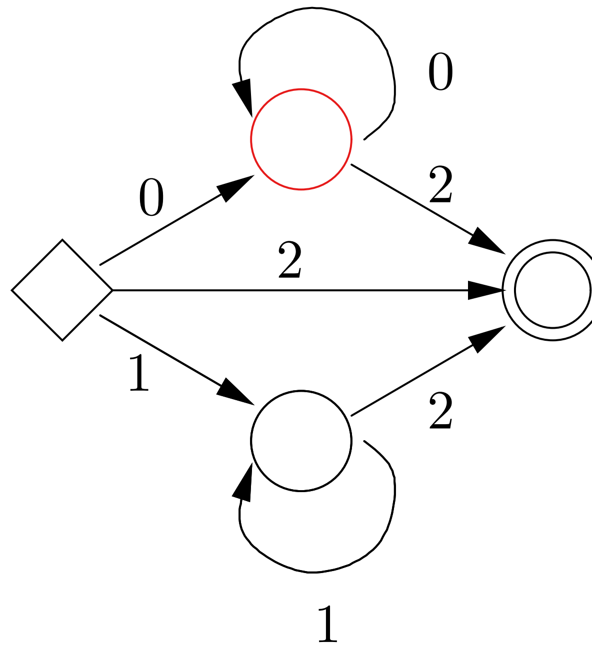
Variation of DFA: Dump states

Strictly speaking, our pictures do not quite fit the definition.



Variation of DFA: Dump states

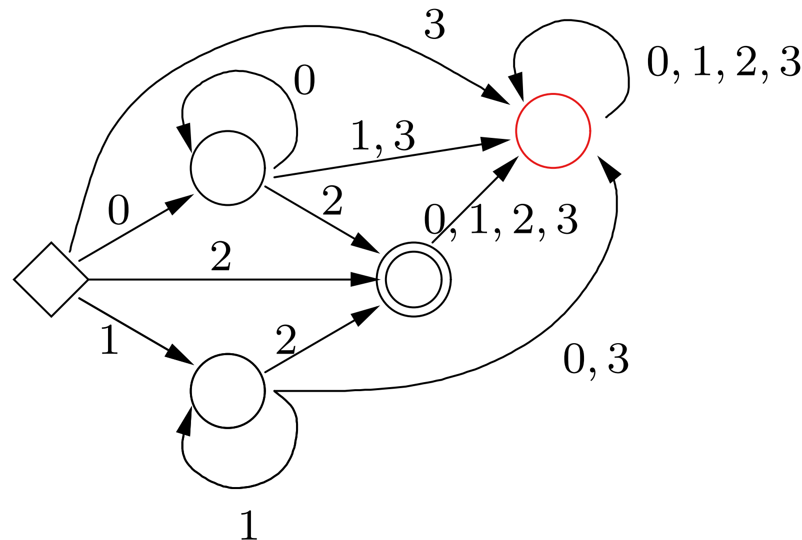
Strictly speaking, our pictures do not quite fit the definition.



There is no arrow labelled 1 from the red state. So reject 001!

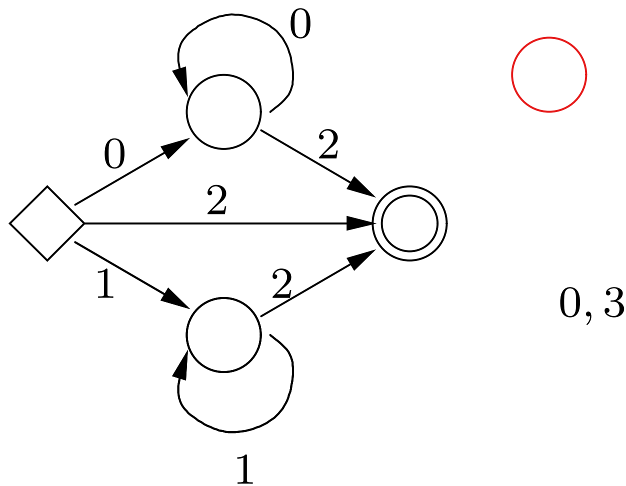
Variation of DFA: Dump states

We do not always draw all the states. We can leave out ‘dump states’; i.e. states from which no accepting state can be reached. If we add a dump state to this DFA, it would look like this:



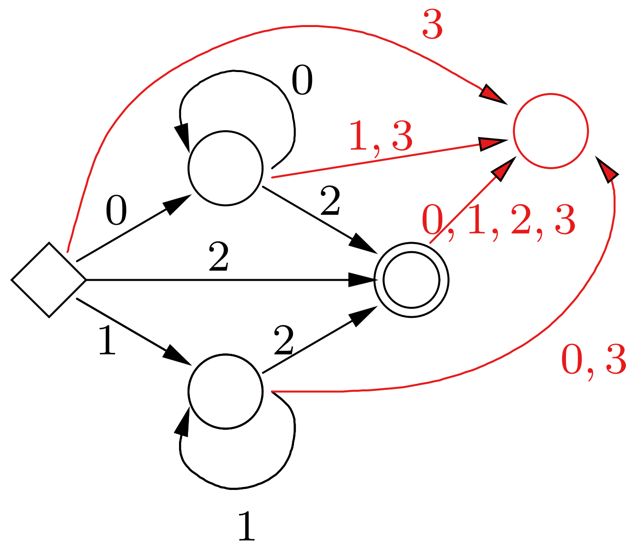
Variation of DFA: Dump states

We obtain this new picture by adding in one new state, . . .



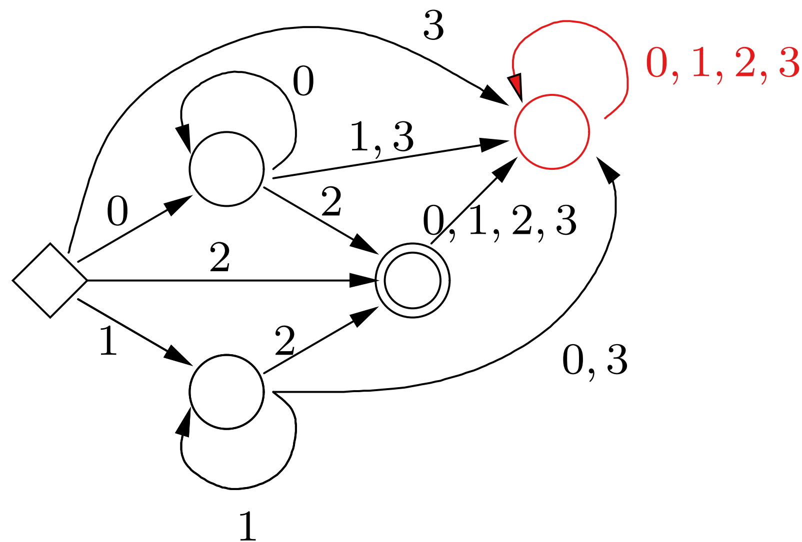
Variation of DFA: Dump states

We obtain this new picture by adding in one new state, and arrows to this new state whenever there is an arrow missing in the original picture.



Variation of DFA: Dump states

Further we add an arrow from the new state to itself labelled with all the letters of the underlying alphabet $\Sigma = \{0, 1, 2, 3\}$.



Variation of DFA: Dump states

If we draw all states, then we can follow a word through the automaton along a unique path without getting stuck. The word is accepted if and only if the state reached when the word ends is an accepting state.

Variation of DFA: Dump states

If we leave out some dump states, then we can follow a word through the automaton along a unique path, but we might get stuck. The word is accepted if and only if we do not get stuck and the state reached when the word ends is an accepting state.

Accepting a word

Definition 9 *A word* $\alpha = x_1 \cdots x_n$ *is accepted by a DFA if*

$$\begin{aligned} \delta(q_\bullet, x_1) &= q_1, \\ \delta(q_1, x_2) &= q_2, \\ &\vdots \\ \delta(q_{n-1}, x_n) &= q_n \in F \end{aligned}$$

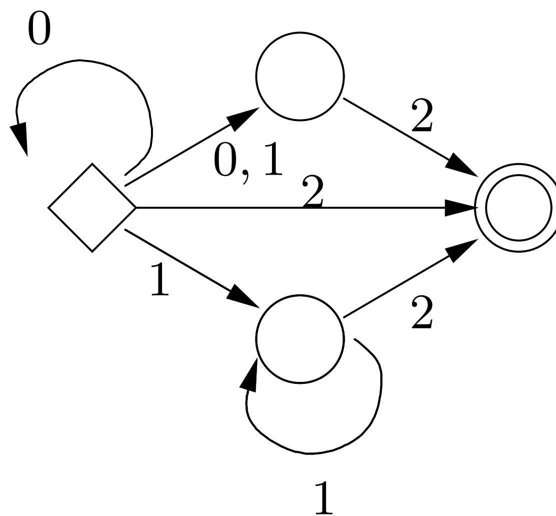
Accepting a word

This means that we have a sequence of states $q_\bullet, q_1, \ldots, q_n$ such that

$$q_\bullet \xrightarrow{x_1} q_1 \xrightarrow{x_2} q_2 \longrightarrow \cdots \xrightarrow{x_n} q_n \in F$$

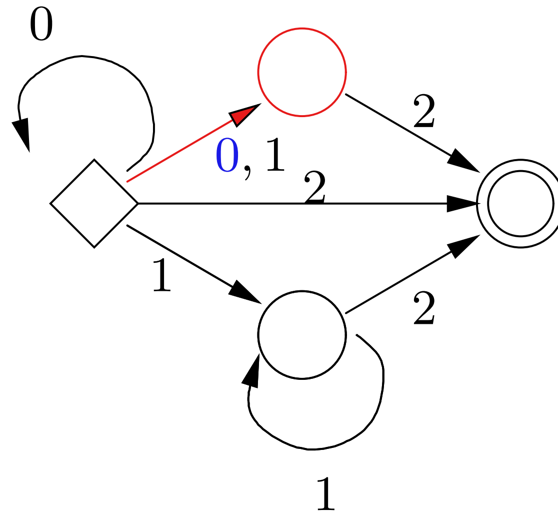
Acceptance for NFAs

When we consider an NFA, there may be more than one way of following a word through the automaton. Consider the word 002.



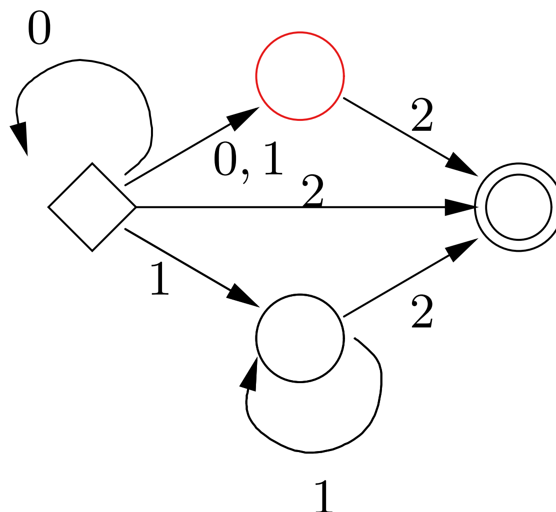
Acceptance for NFAs

First attempt: Consider the word 002.



Acceptance for NFAs

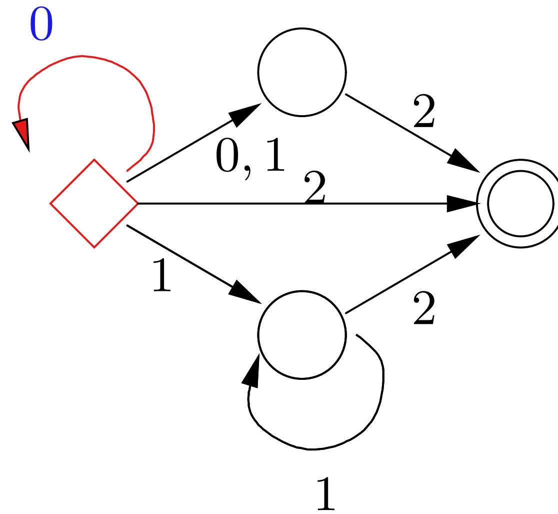
First attempt: Consider the word 002.



We get stuck and this is not an accepting path.

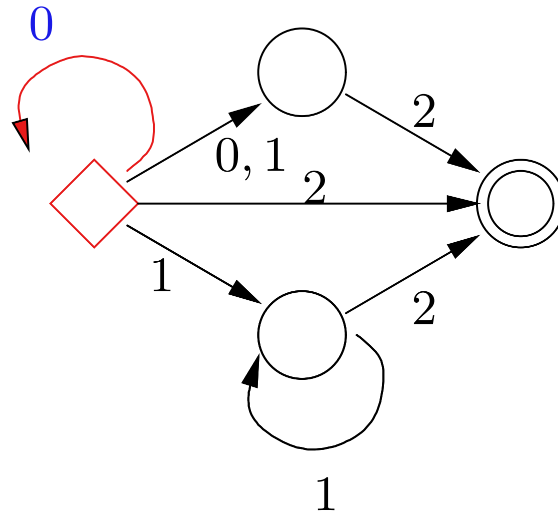
Acceptance for NFAs

Second attempt: Consider the word 002.



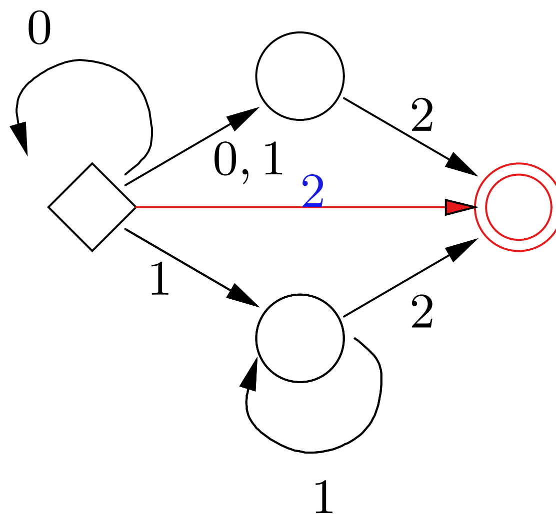
Acceptance for NFAs

Second attempt: Consider the word 002.



Acceptance for NFAs

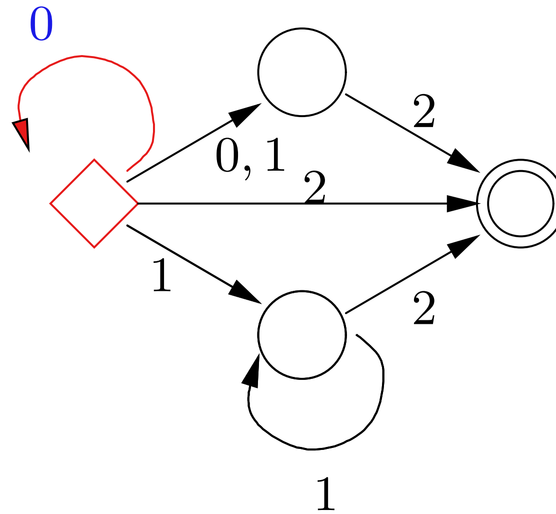
Second attempt: Consider the word 002.



This is an accepting path and the word is accepted.

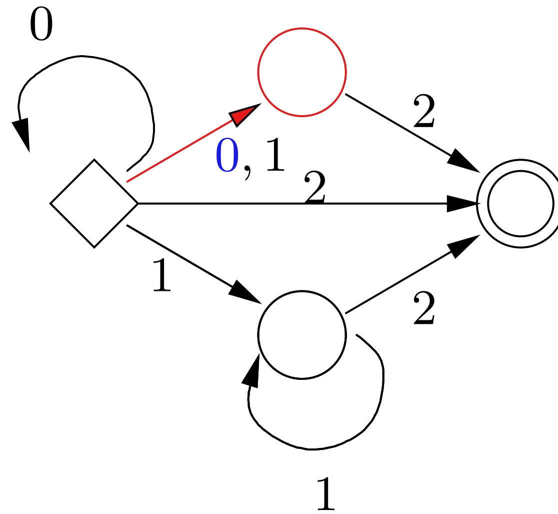
Acceptance for NFAs

Third attempt: Consider the word 002.



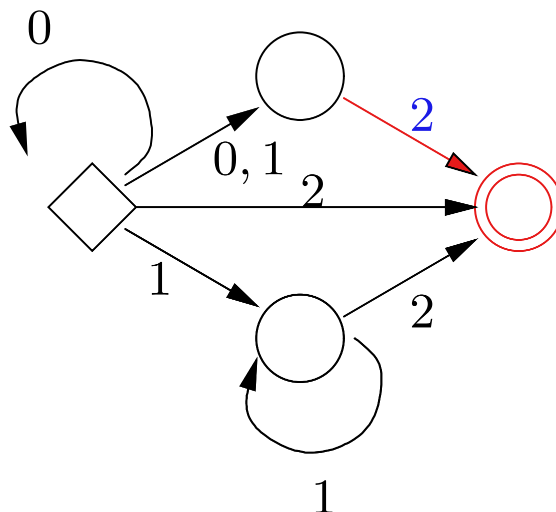
Acceptance for NFAs

Third attempt: Consider the word 002.



Acceptance for NFAs

Third attempt: Consider the word $00\color{blue}{2}$.



This is an accepting path and the word is accepted.

Accepting a word—NFAs

Definition 9 ctd *A word* $\alpha = x_1 \cdots x_n$ *is* *accepted by an NFA* *if there are states*

$$(q_0 = q_\bullet), q_1, \ldots, q_n$$

such that for all $0 \leq i < n$,

$$\delta \text{ relates } (q_i, x_i) \text{ with } q_{i+1}$$

and q_n *is an accepting state.*

Accepting a word—NFAs

In other words, we have a sequence of states $q_\bullet, q_1, \ldots, q_n$ such that

$$q_\bullet \xrightarrow{x_1} q_1 \xrightarrow{x_2} q_2 \longrightarrow \cdots \xrightarrow{x_n} q_n$$

and q_n is an accepting state.

Accepting a word—NFAs

For a DFA, it is easy to decide whether or not a given word is accepted. Just follow the unique path through the automaton given by the word.

Accepting a word—NFAs

For an NFA, there will typically be several paths given by the word. We have to investigate all of those in order to find out whether the word is accepted.

- We will see later why NFAs are nonetheless useful.

A DFA for a regular expression

Proposition 1.2 *For every regular expression over some alphabet* Σ *there is a DFA that accepts precisely those words which match the regular expression.*

A DFA for a regular expression

Proposition 1.2 *For every regular expression over some alphabet* Σ *there is a DFA that accepts precisely those words which match the regular expression.*

However, this isn't as easy as you might first think!

From patterns to DFAs,1

The obvious proof of Proposition 1.2 is to proceed by induction over the way a pattern is built.

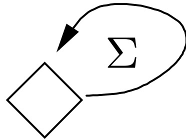
From patterns to DFAs,1

The obvious proof of Proposition 1.2 is to proceed by induction over the way a pattern is built.

And indeed, the start (base cases) looks promising.

From patterns to DFAs,2

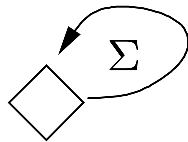
We can define automata for the pattern $\emptyset$ (which matches no word at all).



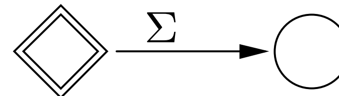
Accepts no word at all

From patterns to DFAs,2

We can define automata for the patterns $\emptyset$ (which matches no word at all), ϵ (which matches precisely the word ϵ.



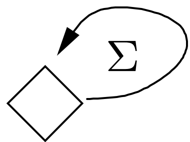
Accepts no word at all



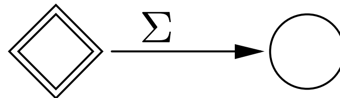
Accepts precisely ϵ

From patterns to DFAs,2

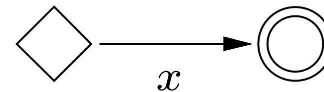
We can define automata for the patterns $\emptyset$ (which matches no word at all), ϵ (which matches precisely the word ϵ and x, which matches precisely the word x, where $x \in \Sigma$.



Accepts no word at all



Accepts precisely ϵ

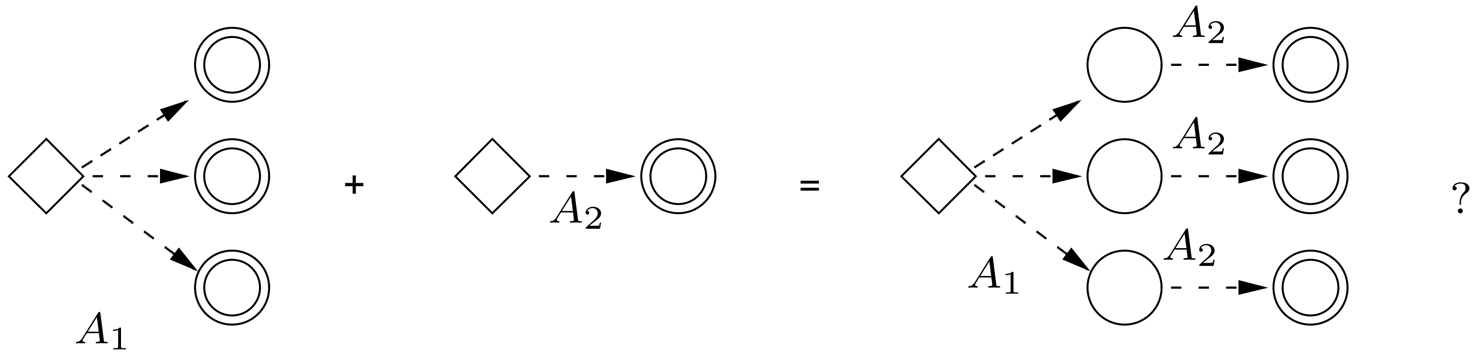


The other forms of pattern

- concatenation: $(p_1 p_2)$
- alternative: $(p_1 | p_2)$
- Kleene star: $p*$

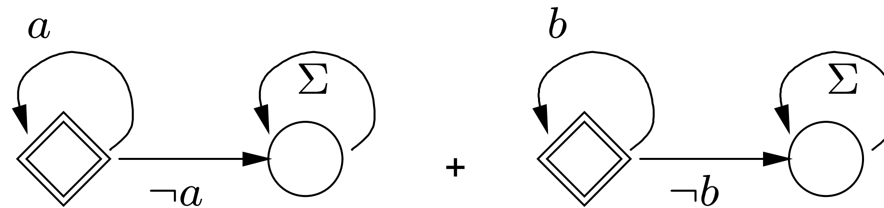
Concatenation

We start running into problems with concatenation:



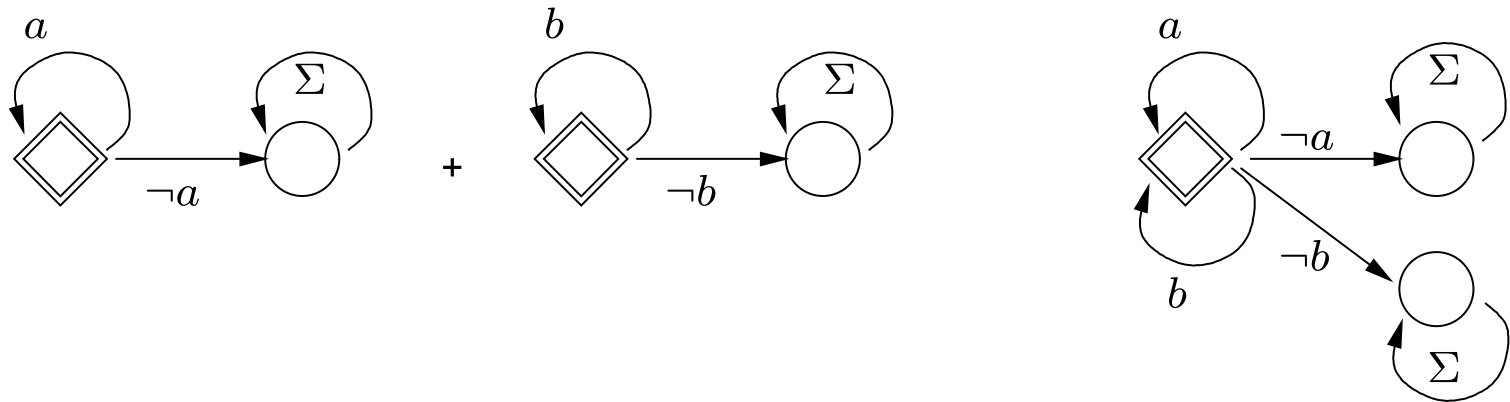
Concatenation

Here is an example which shows why. The pattern $a{*}b{*}$ is the concatenation of $a{*}$ with $b{*}$.



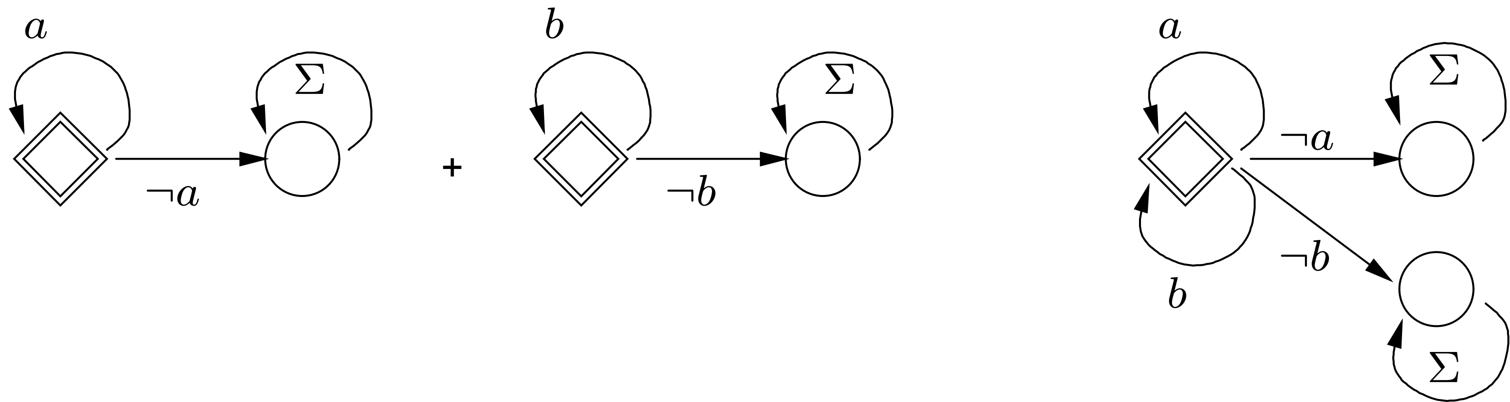
Concatenation

If we treat this according to our original idea, we get



Concatenation

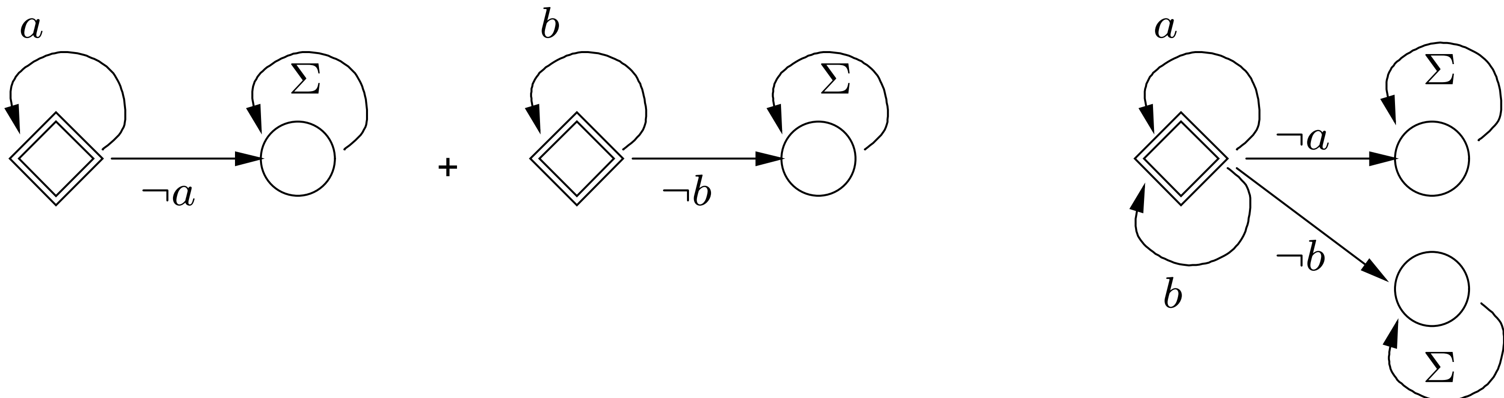
If we treat this according to our original idea, we get



But that allows the string aba to be accepted, which doesn't match $a{*}b{*}$.

τ-transitions,1

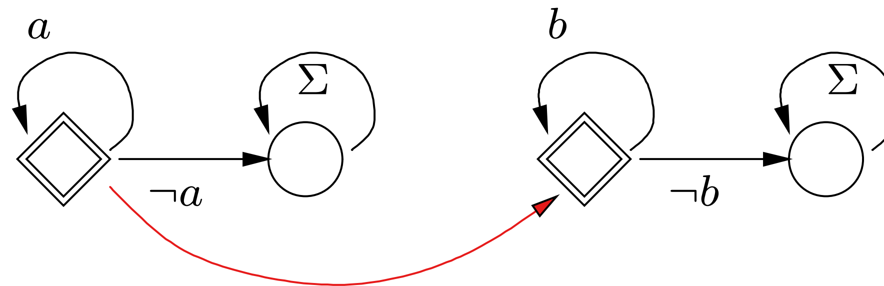
The problem with the example clearly was that we couldn't separate the two automata sufficiently.



τ-transitions,1

What we would have liked to do is

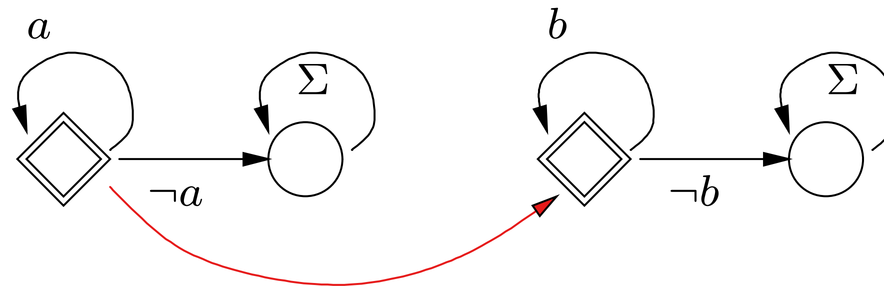
something like this:



τ-transitions,1

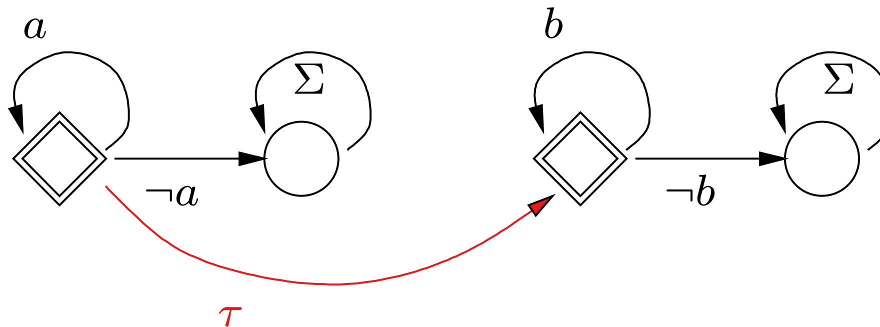
What we would have liked to do is

something like this:



We have drawn an arrow from every accepting state in the first automaton to the start state of the second automaton. Note that this arrow is not labelled!

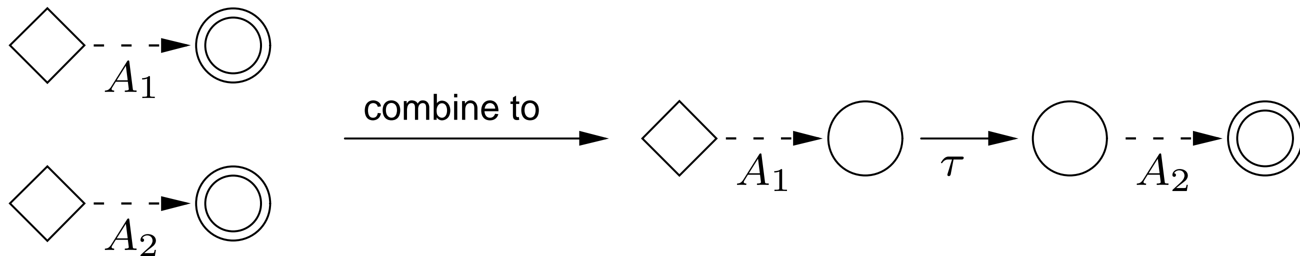
τ-transitions,2



The idea is that we should be able to take this transition whenever we like, without matching it against a letter of the word. We call such transitions τ-transitions and label them in this way to avoid confusion.

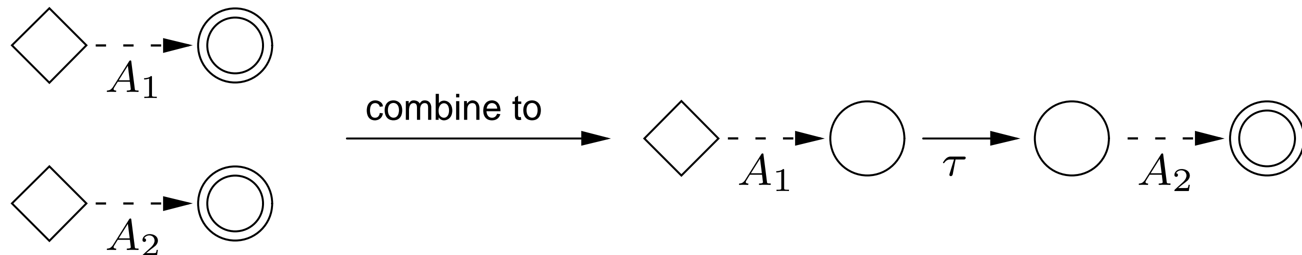
Concatenation revisited

This allows us to deal with concatenation like this.



Concatenation revisited

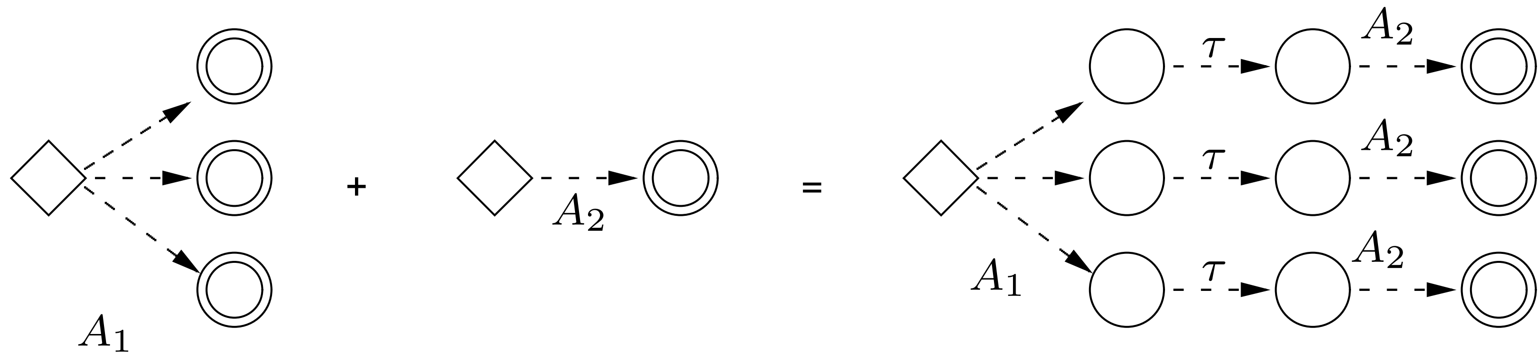
This allows us to deal with concatenation like this.



Note that we need to append the automaton A_2 (via a τ-move) for every accepting state of A_1.

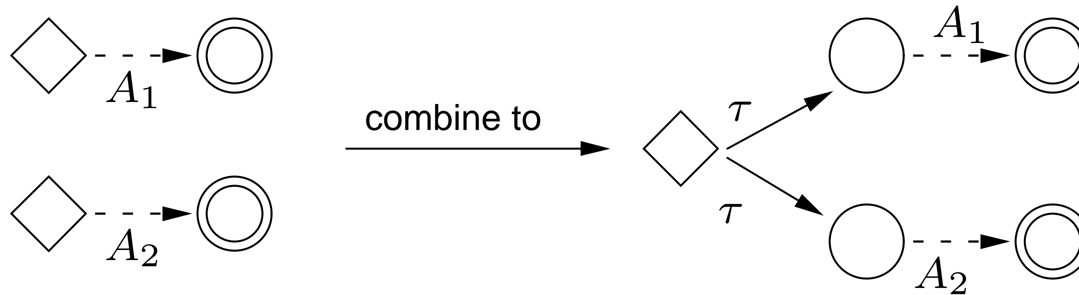
Concatenation revisited

So maybe we should be using this picture instead.



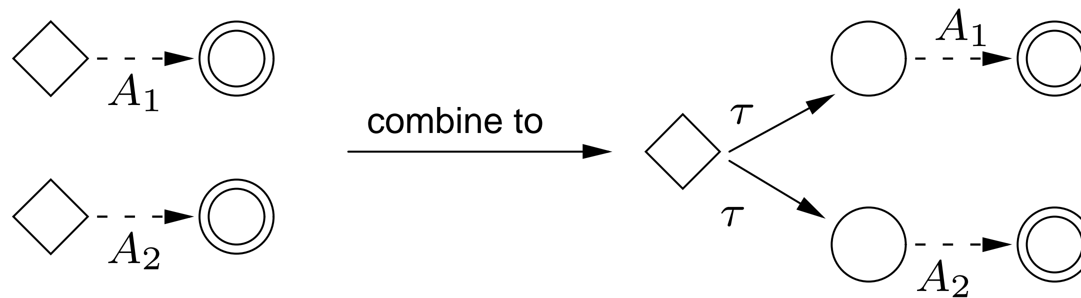
Alternative

Using τ-moves makes it is easy to deal with alternative.



Alternative

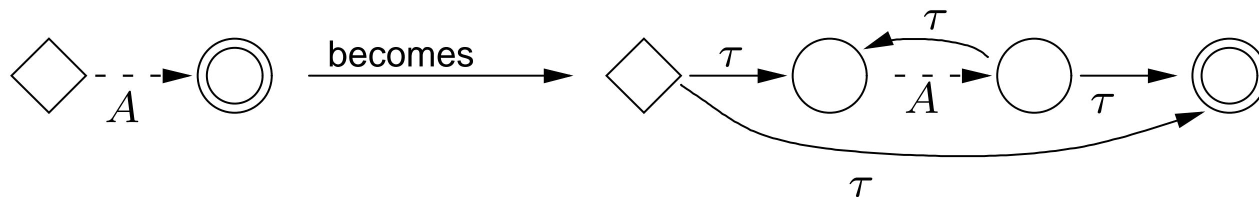
Using τ-moves makes it is easy to deal with alternative.



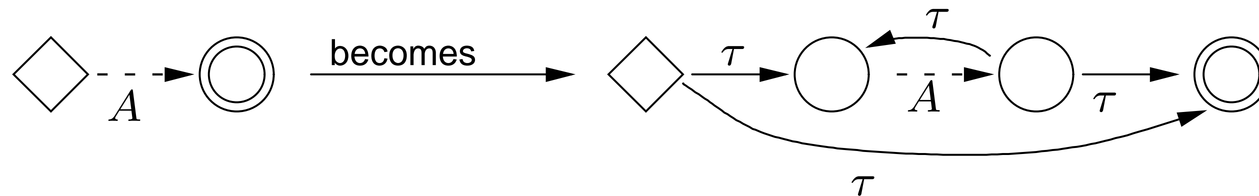
Convince yourself that the new automaton recognizes the intended language!

Kleene star,1

The Kleene star leads to the most complicated looking automaton. This is because to be general enough to apply to all automata, we really do need all these τ-transitions.

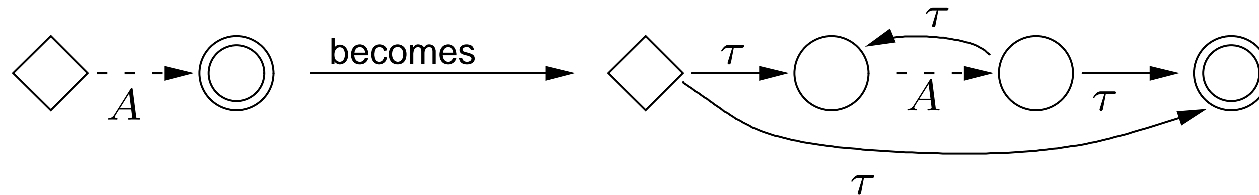


Kleene star,2



Convince yourself that the new automaton recognizes the intended language!

Kleene star,2

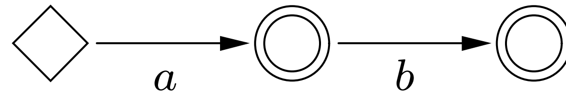


Convince yourself that the new automaton recognizes the intended language!

For many examples we can make do with fewer states and τ-transitions than those given in the general rule!

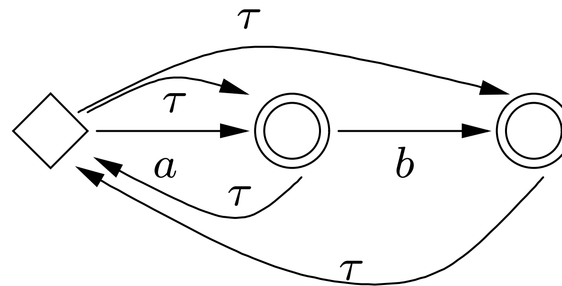
Example 1

Consider the following automaton which corresponds to the pattern $a|ab$.



Example 1

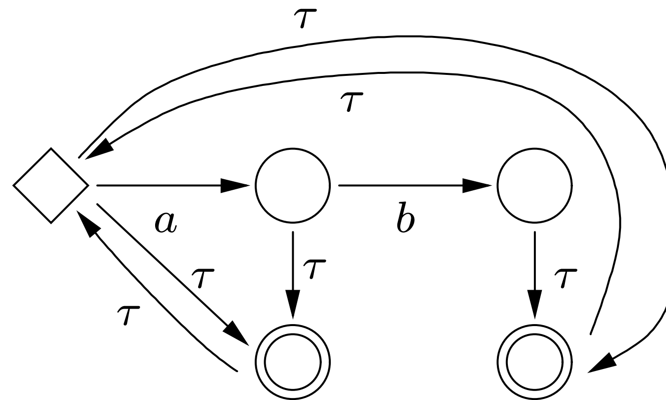
If we just try to insert τ-moves to get an automaton for $(a|ab)*$, we get



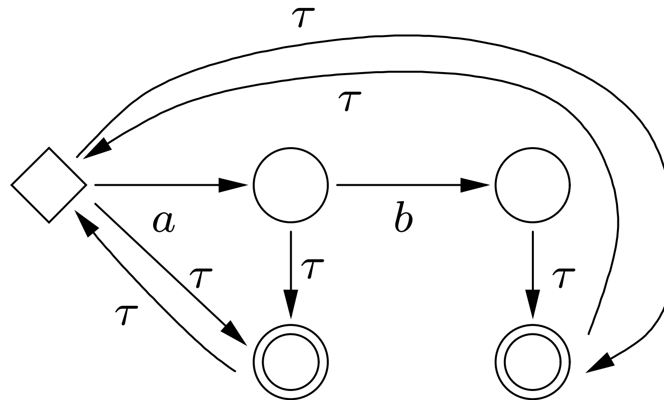
which accepts the word bb, which it shouldn't.

Example 1

Let us now at least deal with the accepting states, as suggested in our rule.



Example 1



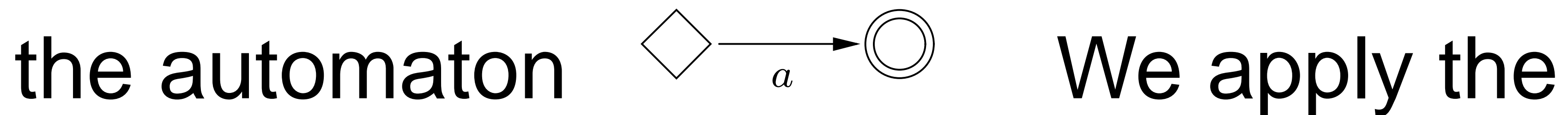
This does work, but if the start state was an accepting state as well, we might have to add another state.

Example 2

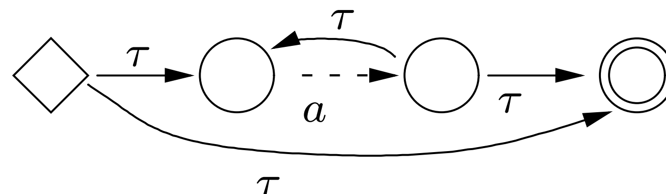
Let us assume we now want to find an NFA with τ-moves for the pattern $a*b*$.

Example 2

Let us assume we now want to find an NFA with τ-moves for the pattern $a{*}b{*}$.

We start with the pattern a, which gives us the automaton 

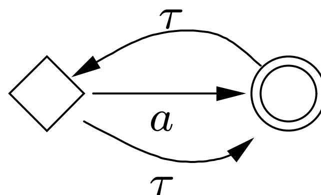
We apply the Kleene star to get an automaton for the pattern $a{*}$.



Example 2

Let us assume we now want to find an NFA with τ-moves for the pattern $a{*}b{*}$.

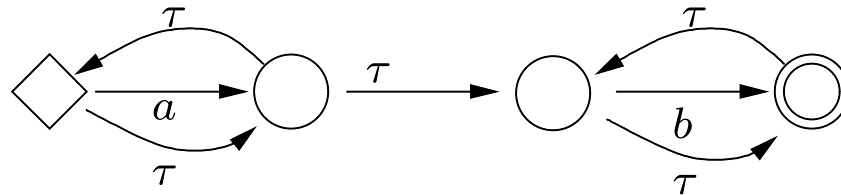
We apply the Kleene star to get an automaton for the pattern $a{*}$. We can remove some superfluous τ-moves.



Example 2

Let us assume we now want to find an NFA with τ-moves for the pattern $a{*}b{*}$.

We get an identical automaton for $b{*}$, and putting the two together we obtain the solution to the overall problem.



We have shown. . .

So what have we done now? We have constructed automata from patterns, but those automata mention a new kind of move (τ-transitions), and they are non-deterministic.

We have shown...

So what have we done now? We have constructed automata from patterns, but those automata mention a new kind of move (τ-transitions), and they are non-deterministic.

In other words we have convinced ourselves that the following holds.

We have shown...

Proposition 1.3 For every regular expression over some alphabet Σ there is an NFA with τ-transitions that accepts precisely those words which match the regular expression.

From NFAs to DFAs,1

Our original aim was, of course, to construct a DFA (without any fancy moves like τ-transitions). We will therefore next concentrate on taking an NFA with such transitions and turning it into a DFA.

From NFAs to DFAs,2

In other words, we want to show the following result.

Proposition 1.4 For every NFA with τ-moves there exists a DFA that accepts precisely the same words.