
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

COMP20121, 'power' part: section 1

lecture 1: Regular languages

lecture 2: Finite automata for regular
languages

LECTURE THREE

Finite automata for regular languages (ctd)

We have shown. . .

What have we done so far? 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. . .

What have we done so far? 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 now 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.*

Getting rid of τ -moves, 1

- We first show that we can turn an NFA with τ -transitions into an ordinary NFA.
- Let $(Q, q_{\bullet}, F, \delta)$ be an NFA with τ -transitions.
- We define a new automaton

$$(Q', q'_{\bullet}, F', \delta')$$

Getting rid of τ -moves, 2

- $Q' = Q,$

Getting rid of τ -moves, 2

- $Q' = Q,$
- $q'_{\bullet} = q_{\bullet},$

Getting rid of τ -moves,2

- $Q' = Q,$
- $q'_{\bullet} = q_{\bullet},$
- F' consists of all states in F as well as all states from which an accepting state can be reached **by only using τ -transitions.**

Getting rid of τ -moves,3

- There is a transition $q \xrightarrow{x} q'$ in δ' if there is a sequence of transitions in δ consisting of an arbitrary number of τ -moves followed by one transition labelled x , that is if

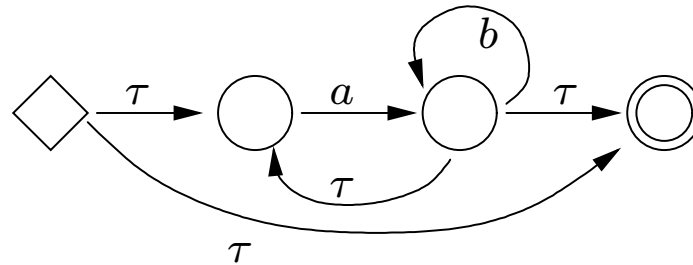
$$q \xrightarrow{\tau} \dots \xrightarrow{\tau} q'' \xrightarrow{x} q'.$$

Getting rid of τ -moves,4

We can typically **clean up** the resulting automaton by removing all states which **cannot be reached from the start state**.

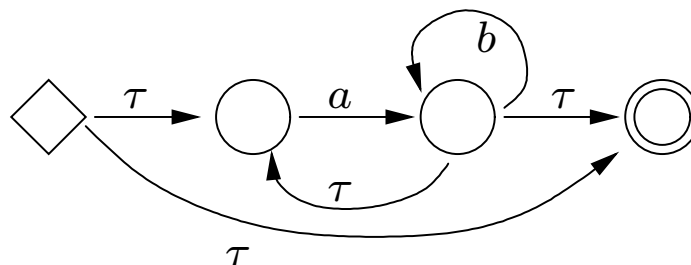
Example

We consider the following automaton.



Example

We consider the following automaton.

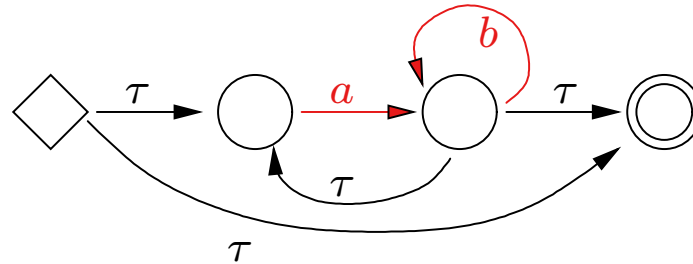


We start by considering just the states.

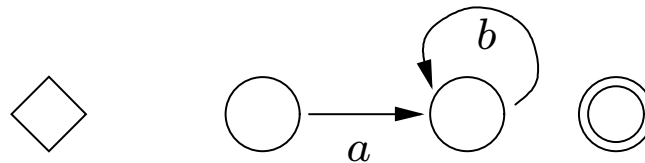


Example

We consider the following automaton.

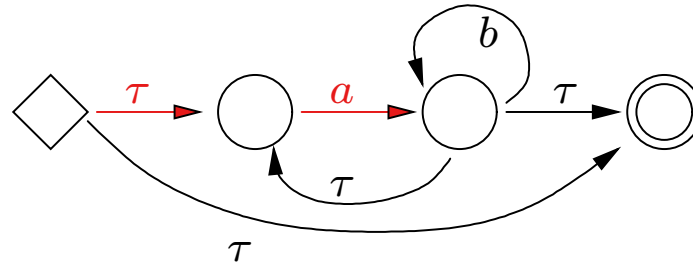


We put in all transitions labelled with letters from Σ (that is, all non- τ -transitions).

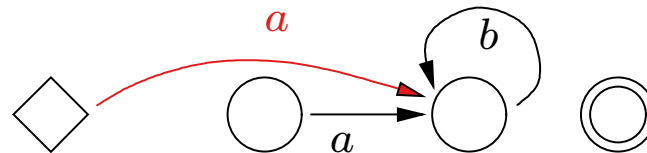


Example

We consider the following automaton.

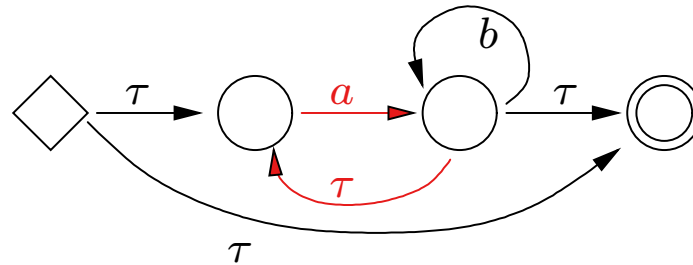


From the start state, we get new transitions

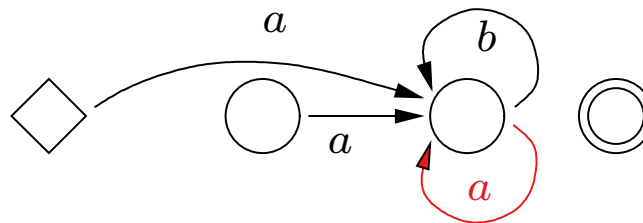


Example

We consider the following automaton.

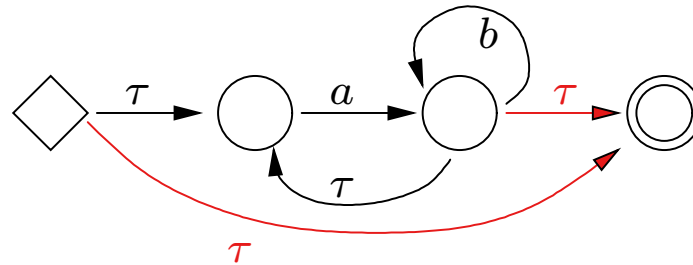


From the next state, we get no new transitions. **From the third state**, we have

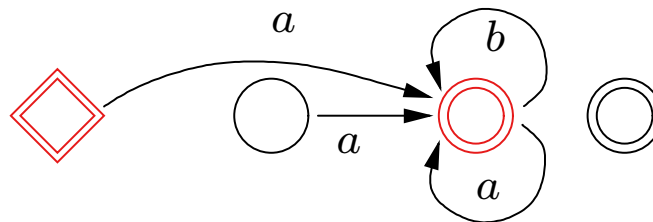


Example

We consider the following automaton.

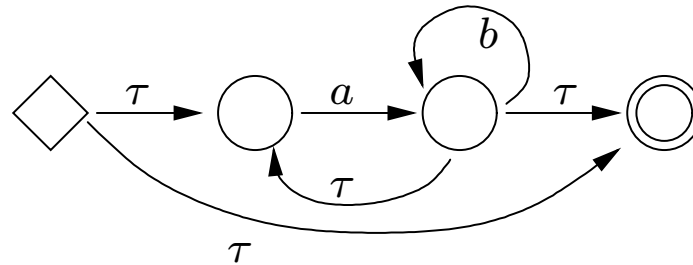


There are no new transitions from the only accepting state. We now add the **new accepting states**.

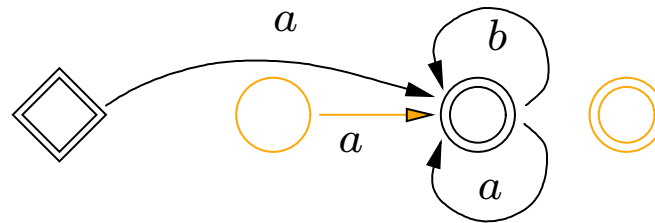


Example

We consider the following automaton.

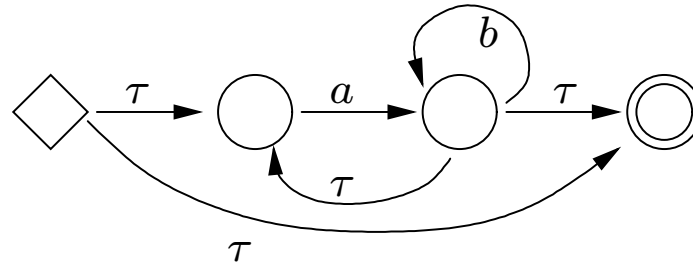


Finally we remove all unreachable states and all transitions involved in those.

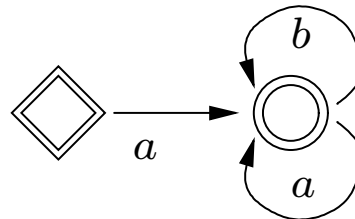


Example

We consider the following automaton.



Finally we remove all unreachable states and all transitions involved in those.



Non-determinacy, 1

- It remains to show that we can turn a non-deterministic automaton into a deterministic one.
- Given an NFA $(Q, q_{\bullet}, F, \delta)$ we define a new automaton

$$(Q', q'_{\bullet}, F', \delta').$$

Non-determinacy,2

- $Q' = \mathcal{P}(Q)$, the set of all subsets of the old set of states.

Non-determinacy,2

- $Q' = \mathcal{P}(Q)$, the set of all subsets of the old set of states.
- $q'_{\bullet} = \{q_{\bullet}\}$, the set containing precisely the old start state.

Non-determinacy,2

- $Q' = \mathcal{P}(Q)$, the set of all subsets of the old set of states.
- $q'_{\bullet} = \{q_{\bullet}\}$, the set containing precisely the old start state.
- F' is the set of all new states which contain at least one old accepting state.

Non-determinacy,3

- To find out to which new state S' a new transition $S \xrightarrow{x} S'$, labelled x , from a new state S will go, calculate

$$S' = \{q' \in Q \mid \exists q \in S. \text{ in } \delta, q \xrightarrow{x} q'\}.$$

Non-determinacy,4

$$S' = \{q' \in Q \mid \exists q \in S. \text{ in } \delta, q \xrightarrow{x} q'\}$$

In other words, for each state $q \in S$, take the states q' for which you have $q \xrightarrow{x} q'$. Collect all these together and call the result S' .

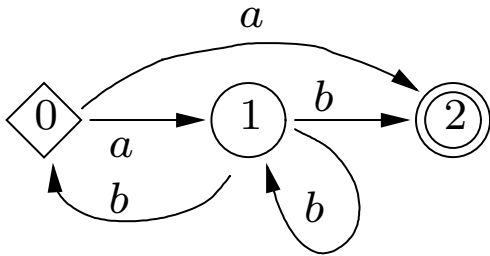
Non-determinacy,4

$$S' = \{q' \in Q \mid \exists q \in S. \text{ in } \delta, q \xrightarrow{x} q'\}$$

In other words, for each state $q \in S$, take the states q' for which you have $q \xrightarrow{x} q'$. Collect all these together and call the result S' . Again we can simplify our picture a lot if we remove unreachable states.

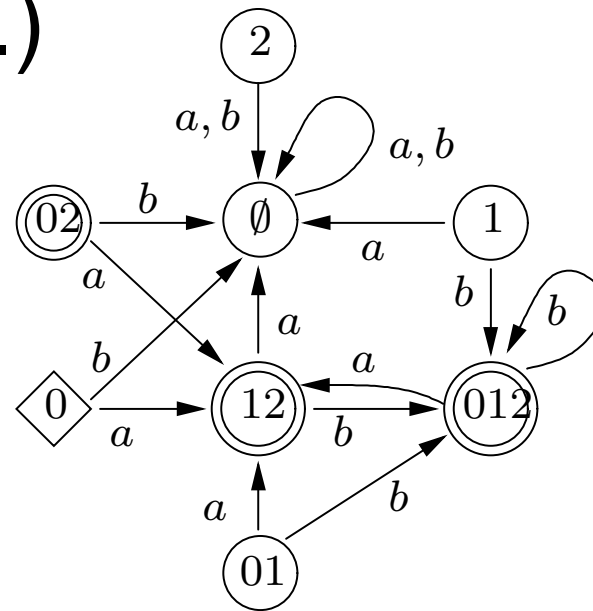
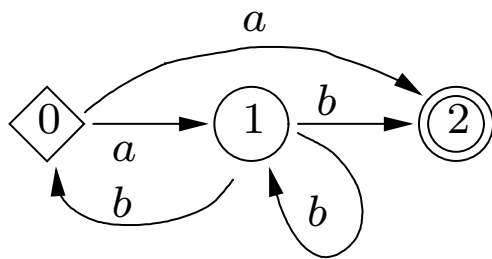
Example

We look at the following NFA.



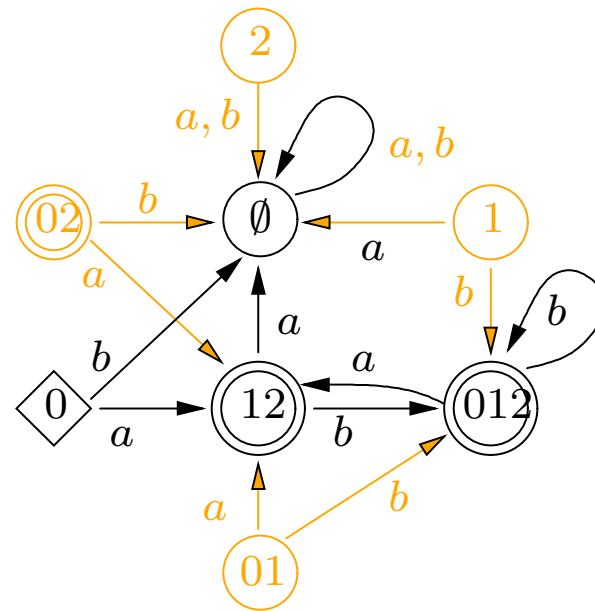
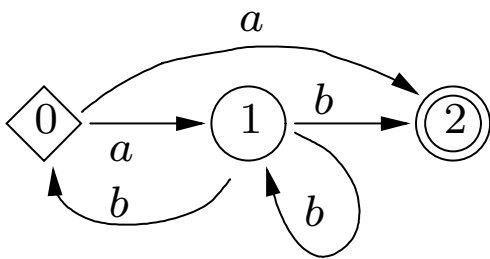
Example

When we draw **all** the states, the result is quite large! (In fact, it will be exponential in the number of states.)



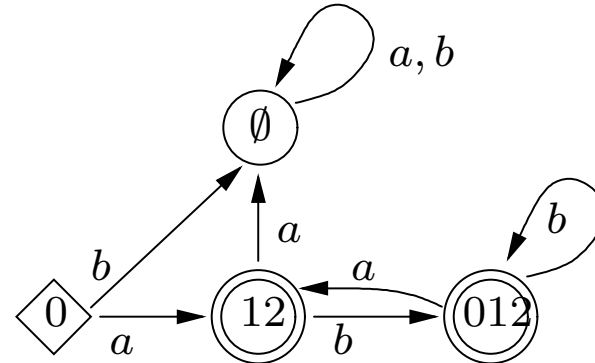
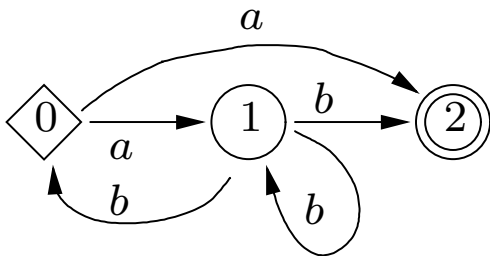
Example

But when we remove the **unreachable** states and all transitions they involve, it becomes clear that we don't need most of the picture!



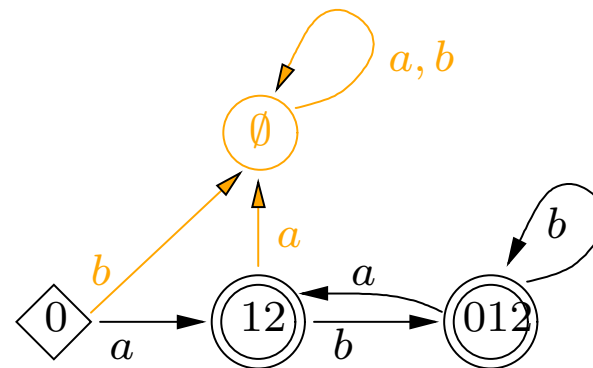
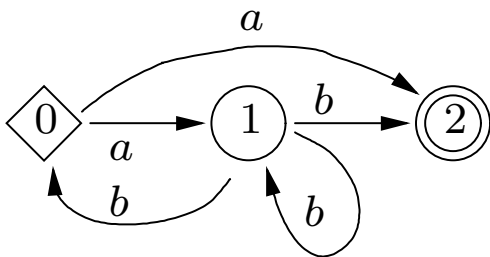
Example

But when we remove the unreachable states, it becomes clear that we don't need most of the picture!



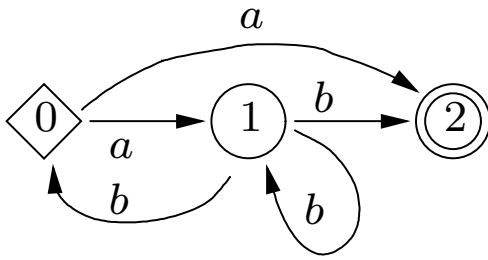
Example

It also is clear that \emptyset will always be a dump state (the only transitions from \emptyset have to be to \emptyset , and it is never an accepting state, so we can leave that out as well if we like).



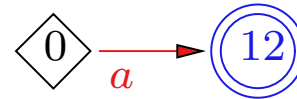
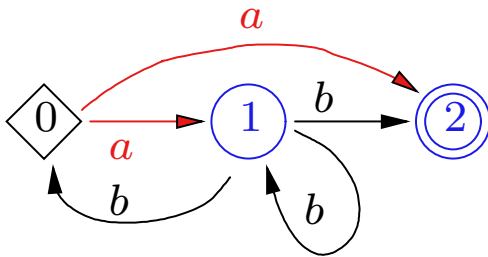
Example

The clever thing to do is to generate the reachable states only in the first place. We start with the new start state, $\{0\}$.



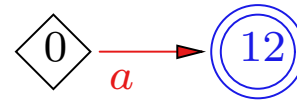
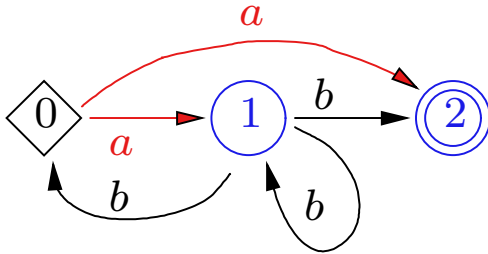
Example

The clever thing to do is to generate the reachable states only. From 0 with an a we can go to 1 and 2.



From 0 with a b we can go nowhere, so there is no other transition from $\{0\}$.

Example

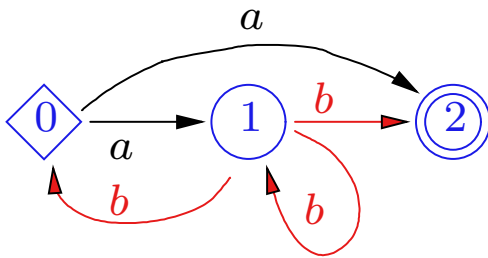


From 1 with an a we can go nowhere, nor can we go anywhere from 2 with an a .

Hence there is no transition labelled a from $\{1, 2\}$.

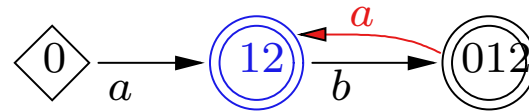
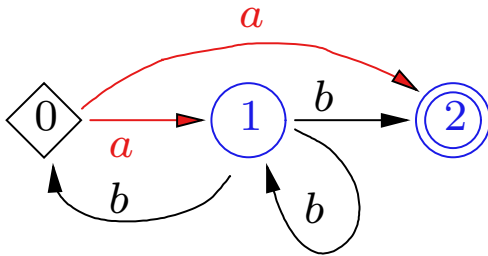
Example

But from 1 with a b we can go to 0, 1 and 2 (but nowhere from 2), so from $\{1, 2\}$ there is a transition labelled b to $\{0, 1, 2\}$.



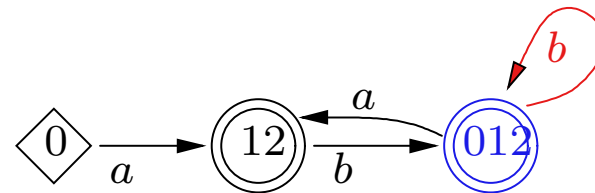
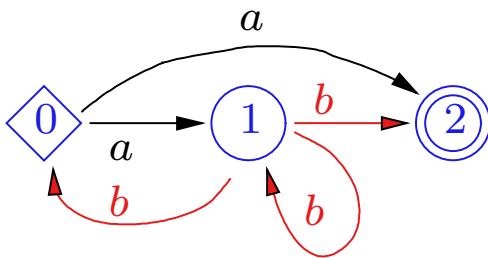
Example

From the states in $\{0, 1, 2\}$ with an a we can go to 1 and 2, so from $\{0, 1, 2\}$ there is a transition labelled a to $\{1, 2\}$.



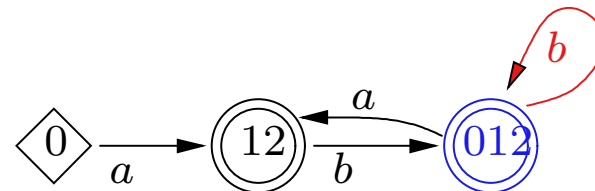
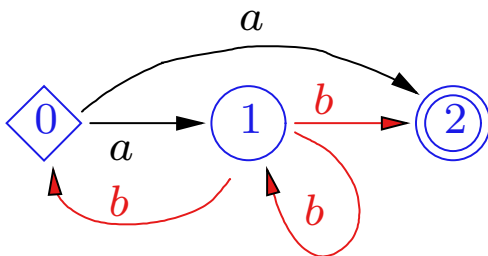
Example

From the states in $\{0, 1, 2\}$ with a b we can go to 0, 1 and 2, so from $\{0, 1, 2\}$ there is a transition labelled b to itself.



Example

From the states in $\{0, 1, 2\}$ with a b we can go to 0, 1 and 2, so from $\{0, 1, 2\}$ there is a transition labelled b to itself.



This is certainly much easier than drawing all the states!

Result

Summarizing our efforts so far, we have demonstrated the following result.

Theorem *Every regular language is recognised by a DFA.*

Along the way we showed that if a language is recognised by an NFA then it is also recognised by a DFA, so that NFAs are no more powerful than DFAs.

NFA Languages are regular

We now prove the converse to the previous result:

Theorem *Every language recognised by a DFA (or even an NFA) is a regular language.*

NFA Languages are regular

We now prove the converse to the previous result:

Theorem *Every language recognised by a DFA (or even an NFA) is a regular language.*

So finite automata are no more powerful in describing languages than patterns.

Accepting paths

To find out which words are accepted by an automaton, we have to find all the words that label the paths from the start state to any of the accepting states.

Paths and their labels

Let (Q, q_1, F, δ) be an NFA, with set $Q = \{q_1, \dots, q_n\}$ of states.

Paths and their labels

Let (Q, q_1, F, δ) be an NFA, with set $Q = \{q_1, \dots, q_n\}$ of states.

- A path of length m from q_i to q_j :

$$q_i \xrightarrow{x_1} q_{j_1} \xrightarrow{x_2} \dots \xrightarrow{x_{m-1}} q_{j_{m-1}} \xrightarrow{x_m} q_j$$

- $x_1 x_2 \dots x_{m-1} x_m$ labels the path.

Paths and their labels

Let (Q, q_1, F, δ) be an NFA, with set $Q = \{q_1, \dots, q_n\}$ of states.

- A path of length m from q_i to q_j :

$$q_i \xrightarrow{x_1} q_{j_1} \xrightarrow{x_2} \dots \xrightarrow{x_{m-1}} q_{j_{m-1}} \xrightarrow{x_m} q_j$$

- $x_1 x_2 \dots x_{m-1} x_m$ labels the path.
- $q_{j_1}, \dots, q_{j_{m-1}}$ are the interior nodes.

Special paths

$$Q_i \xrightarrow{x_1} Q_{j_1} \xrightarrow{x_2} \dots \xrightarrow{x_{m-1}} Q_{j_{m-1}} \xrightarrow{x_m} Q_j$$

Special paths

$$q_i \xrightarrow{x_1} q_{j_1} \xrightarrow{x_2} \cdots \xrightarrow{x_{m-1}} q_{j_{m-1}} \xrightarrow{x_m} q_j$$

- Allow $m = 0$. Then $i = j$ and the path is just q_i and has the label ϵ .

Special paths

$$q_i \xrightarrow{x_1} q_{j_1} \xrightarrow{x_2} \cdots \xrightarrow{x_{m-1}} q_{j_{m-1}} \xrightarrow{x_m} q_j$$

- Allow $m = 0$. Then $i = j$ and the path is just q_i and has the label ϵ .

- Allow $m = 1$. Then the path is just $q_i \xrightarrow{x_1} q_j$ and has the label x_1 .

Special paths

$$q_i \xrightarrow{x_1} q_{j_1} \xrightarrow{x_2} \cdots \xrightarrow{x_{m-1}} q_{j_{m-1}} \xrightarrow{x_m} q_j$$

- Allow $m = 0$. Then $i = j$ and the path is just q_i and has the label ϵ .
- Allow $m = 1$. Then the path is just $q_i \xrightarrow{x_1} q_j$ and has the label x_1 .
- When $m = 0$ or $m = 1$ there are no interior nodes.

The problem

- $P_{i \rightarrow j}$ is the set of paths from q_i to q_j .

The problem

- $P_{i \rightarrow j}$ is the set of paths from q_i to q_j .
- $L_{i \rightarrow j}$ is the language consisting of the words that label the paths in $P_{i \rightarrow j}$.

The problem

- $P_{i \rightarrow j}$ is the set of paths from q_i to q_j .
- $L_{i \rightarrow j}$ is the language consisting of the words that label the paths in $P_{i \rightarrow j}$.
- So we want to determine

$$L_{1 \rightarrow l_1} \cup L_{1 \rightarrow l_2} \cup \dots$$

where q_{l_1}, q_{l_2}, \dots are the accepting states.

Restricted paths

For $k = 0, 1, \dots, n$

- $P_{i \rightarrow j}^k$ is the set of paths from q_i to q_j all of whose interior nodes are in

$$\{q_1, \dots, q_k\}.$$

Restricted paths

For $k = 0, 1, \dots, n$

- $P_{i \rightarrow j}^k$ is the set of paths from q_i to q_j all of whose interior nodes are in

$$\{q_1, \dots, q_k\}.$$

- $L_{i \rightarrow j}^k$ is the language of words that label paths in $P_{i \rightarrow j}^k$.

Note: $L_{i \rightarrow j} = L_{i \rightarrow j}^n$.

What paths are in $P_{i \rightarrow j}^0$?

- All paths $q_i \xrightarrow{x} q_j$ of length 1.
- Also the path q_i of length 0, when $i = j$.

What paths are in $P_{i \rightarrow j}^0$?

- All paths $q_i \xrightarrow{x} q_j$ of length 1.
- Also the path q_i of length 0, when $i = j$.

So $L_{i \rightarrow j}^0$ is the language consisting of the letters x labelling paths $q_i \xrightarrow{x} q_j$ of length 1, and also the word ϵ in the case when $i = j$.

What paths are in $P_{i \rightarrow j}^m$ ($m > 0$)?

- All paths composed of

$q_i \rightarrow \dots \rightarrow q_m$ a path in $P_{i \rightarrow m}^{m-1}$

$q_m \rightarrow \dots \rightarrow q_m$ zero, one

⋮

or more

$q_m \rightarrow \dots \rightarrow q_m$ paths in $P_{m \rightarrow m}^{m-1}$

$q_m \rightarrow \dots \rightarrow q_j$ a path in $P_{m \rightarrow j}^{m-1}$

- Also all paths in $P_{i \rightarrow j}^{m-1}$.

What words are in $L_{i \rightarrow j}^m$ ($m > 0$)?

- So any word in $L_{i \rightarrow j}^m$ must either be in $L_{i \rightarrow j}^{m-1}$ or else must have the form

$$\alpha_0 \alpha_1 \cdots \alpha_k \alpha_{k+1}$$

where

$$\alpha_0 \in L_{i \rightarrow m}^{m-1}, \quad \alpha_1, \dots, \alpha_k \in L_{m \rightarrow m}^{m-1}$$

and $\alpha_{k+1} \in L_{m \rightarrow j}^{m-1}$.

What words are in $L_{i \rightarrow j}^m$ ($m > 0$)?

- All words in $L_{i \rightarrow j}^{m-1}$ and
- all words $\alpha_0(\alpha_1 \cdots \alpha_k)\alpha_{k+1}$ in

$$L_{i \rightarrow m}^{m-1} \cdot (L_{m \rightarrow m}^{m-1})^* \cdot L_{m \rightarrow j}^{m-1}$$

What words are in $L_{i \rightarrow j}^m$ ($m > 0$)?

- All words in $L_{i \rightarrow j}^{m-1}$ and
- all words $\alpha_0(\alpha_1 \cdots \alpha_k)\alpha_{k+1}$ in

$$L_{i \rightarrow m}^{m-1} \cdot (L_{m \rightarrow m}^{m-1})^* \cdot L_{m \rightarrow j}^{m-1}$$

So

$$L_{i \rightarrow j}^m = L_{i \rightarrow j}^{m-1} \cup (L_{i \rightarrow m}^{m-1} \cdot L_{m \rightarrow m}^{m-1*} \cdot L_{m \rightarrow j}^{m-1}).$$

Special Cases

$$L_{i \rightarrow j}^m = L_{i \rightarrow j}^{m-1} \cup (L_{i \rightarrow m}^{m-1} \cdot L_{m \rightarrow m}^{m-1*} \cdot L_{m \rightarrow j}^{m-1}).$$

- If $m = j$ then

$$L_{i \rightarrow j}^m = L_{i \rightarrow m}^{m-1} \cdot (L_{m \rightarrow m}^{m-1})^*.$$

Special Cases

$$L_{i \rightarrow j}^m = L_{i \rightarrow j}^{m-1} \cup (L_{i \rightarrow m}^{m-1} \cdot L_{m \rightarrow m}^{m-1*} \cdot L_{m \rightarrow j}^{m-1}).$$

- If $m = j$ then

$$L_{i \rightarrow j}^m = L_{i \rightarrow m}^{m-1} \cdot (L_{m \rightarrow m}^{m-1})^*.$$

- If $m = i$ then

$$L_{i \rightarrow j}^m = (L_{m \rightarrow m}^{m-1})^* \cdot L_{m \rightarrow j}^{m-1}.$$

Unravelling

The formula

$$L_{i \rightarrow j}^m = L_{i \rightarrow j}^{m-1} \cup (L_{i \rightarrow m}^{m-1} \cdot L_{m \rightarrow m}^{m-1*} \cdot L_{m \rightarrow j}^{m-1}).$$

allows us to ‘unravel’ any $L_{i \rightarrow j}^n$.

Unravelling

The formula

$$L_{i \rightarrow j}^m = L_{i \rightarrow j}^{m-1} \cup (L_{i \rightarrow m}^{m-1} \cdot L_{m \rightarrow m}^{m-1*} \cdot L_{m \rightarrow j}^{m-1}).$$

allows us to ‘unravel’ any $L_{i \rightarrow j}^n$.

We do this by reducing the superscript n *via* $n - 1, n - 2, \dots$, to 1 and finally to 0. Then the $L_{i \rightarrow j}^0$ are given directly.

Unravelling

Often we can read off the desired language before going all the way down to index 0, which is always recommended! However, we are describing an algorithm which can be mechanized, that is programmed, and computers can't just 'read off' anything.

Simplifying it, 1

When we have finished the unravelling process we are left with an expression containing sets of letters, \cup , $(-)^*$ and \cdot .

This we can simplify according to the following rules:

Simplifying it,2

- $\emptyset \cup L = L = L \cup \emptyset;$

Simplifying it,2

- $\emptyset \cup L = L = L \cup \emptyset;$
- $\emptyset^* = \{\epsilon\};$

Simplifying it,2

- $\emptyset \cup L = L = L \cup \emptyset;$
- $\emptyset^* = \{\epsilon\};$
- $\emptyset \cdot L = \emptyset = L \cdot \emptyset;$

Simplifying it,2

- $\emptyset \cup L = L = L \cup \emptyset;$
- $\emptyset^* = \{\epsilon\};$
- $\emptyset \cdot L = \emptyset = L \cdot \emptyset;$
- $(\{\epsilon\} \cup L)^* = L^* = (L \cup \{\epsilon\})^*;$

Simplifying it,2

- $\emptyset \cup L = L = L \cup \emptyset;$
- $\emptyset^* = \{\epsilon\};$
- $\emptyset \cdot L = \emptyset = L \cdot \emptyset;$
- $(\{\epsilon\} \cup L)^* = L^* = (L \cup \{\epsilon\})^*;$
- $\{\epsilon\}^* = \{\epsilon\};$

Turning it into a pattern, 1

When we have finished the unravelling process we are left with an expression containing sets of letters or ϵ , \cup , $(-)^*$ and \cdot . This we can turn into a pattern as follows.

Turning it into a pattern,2

- For a set consisting of the letters

$x_1, x_2 \dots, x_m$ use $(x_1|x_2|\dots|x_m)$;

Turning it into a pattern,2

- For a set consisting of the letters

$x_1, x_2 \dots, x_m$ use $(x_1|x_2|\dots|x_m)$;

- for \cup use $|$;
- for $\{\epsilon\}$ use ϵ ;

Turning it into a pattern,2

- For a set consisting of the letters

$x_1, x_2 \dots, x_m$ use $(x_1|x_2|\dots|x_m)$;

- for \cup use $|$; ● for $\{\epsilon\}$ use ϵ ;

- for $(-)^*$ use $(-)^*$;

Turning it into a pattern,2

- For a set consisting of the letters

$x_1, x_2 \dots, x_m$ use $(x_1|x_2|\dots|x_m)$;

- for \cup use $|$; ● for $\{\epsilon\}$ use ϵ ;

- for $(-)^*$ use $(-)^*$;

- for \cdot use nothing; ● for \emptyset use \emptyset .

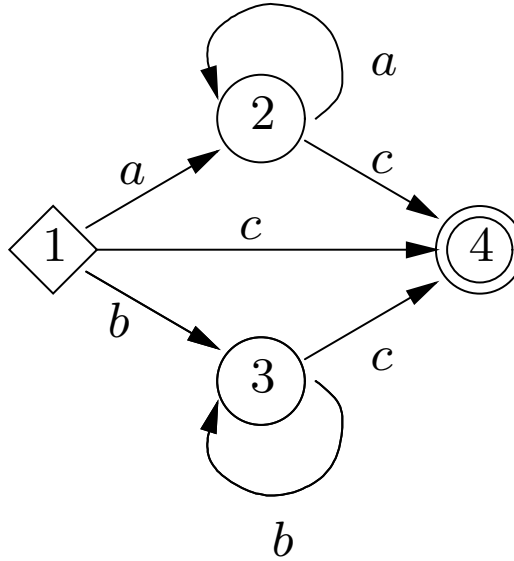
Turning it into a pattern,2

- For a set consisting of the letters $x_1, x_2 \dots, x_m$ use $(x_1|x_2|\dots|x_m)$;
- for \cup use $|$; ● for $\{\epsilon\}$ use ϵ ;
- for $(-)^*$ use $(-)^*$;
- for \cdot use nothing; ● for \emptyset use \emptyset .

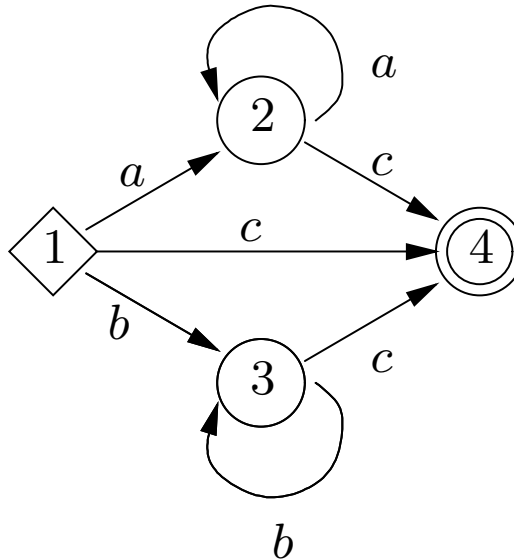
This results in a regular expression.

An example

Consider the following DFA.



An example



The start state is 1, the only accepting state 4, so we are interested in $L_{1 \rightarrow 4}^4$.

An example (continued)

We apply the special case formula

$L_{i \rightarrow m}^m = (L_{i \rightarrow m}^{m-1} \cdot L_{m \rightarrow m}^{m-1*})$ to calculate

$$L_{1 \rightarrow 4}^4 = L_{1 \rightarrow 4}^3 \cdot L_{4 \rightarrow 4}^{3*}$$

An example (continued)

We apply the special case formula

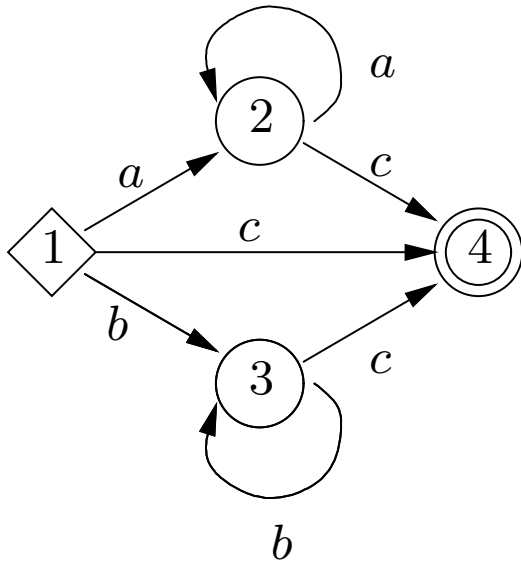
$$L_{i \rightarrow m}^m = (L_{i \rightarrow m}^{m-1} \cdot L_{m \rightarrow m}^{m-1*}) \text{ to calculate}$$

$$L_{1 \rightarrow 4}^4 = L_{1 \rightarrow 4}^3 \cdot L_{4 \rightarrow 4}^3^*$$

We apply the general formula to get

$$L_{1 \rightarrow 4}^3 = L_{1 \rightarrow 4}^2 \cup (L_{1 \rightarrow 3}^2 \cdot L_{3 \rightarrow 3}^2^* \cdot L_{3 \rightarrow 4}^2)$$

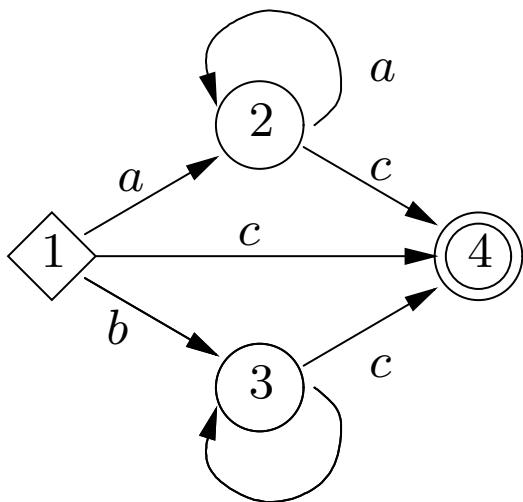
An example (continued)



Also $L_{4 \rightarrow 4}^3 = \{\epsilon\}$

(since there is no way of going from 4 to 4). So

An example (continued)

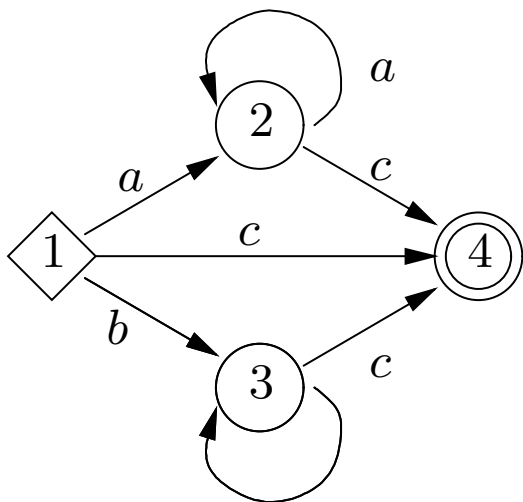


Also $L_{4 \rightarrow 4}^3 = \{\epsilon\}$

(since there is no way of going from 4 to 4). So

$$\begin{aligned} L_{1 \rightarrow 4}^4 &= L_{1 \rightarrow 4}^3 \cdot \{\epsilon\}^* = L_{1 \rightarrow 4}^3 \\ &= L_{1 \rightarrow 4}^2 \cup (L_{1 \rightarrow 3}^2 \cdot L_{3 \rightarrow 3}^2{}^* \cdot L_{3 \rightarrow 4}^2) \end{aligned}$$

An example (continued)



Also $L_{4 \rightarrow 4}^3 = \{\epsilon\}$

(since there is no way of going from 4 to 4). So

$$\begin{aligned} L_{1 \rightarrow 4}^4 &= L_{1 \rightarrow 4}^3 \cdot \{\epsilon\}^* = L_{1 \rightarrow 4}^3 \\ &= L_{1 \rightarrow 4}^2 \cup (L_{1 \rightarrow 3}^2 \cdot L_{3 \rightarrow 3}^2 \cdot L_{3 \rightarrow 4}^2) \end{aligned}$$

Also $L_{3 \rightarrow 3}^2 = \{b, \epsilon\}$ and $L_{3 \rightarrow 4}^2 = \{c\}$.

An example (continued)

So

$$L_{1 \rightarrow 4}^4 = L_{1 \rightarrow 4}^2 \cup (L_{1 \rightarrow 3}^2 \cdot \{b, \epsilon\}^* \cdot \{c\})$$

An example (continued)

So

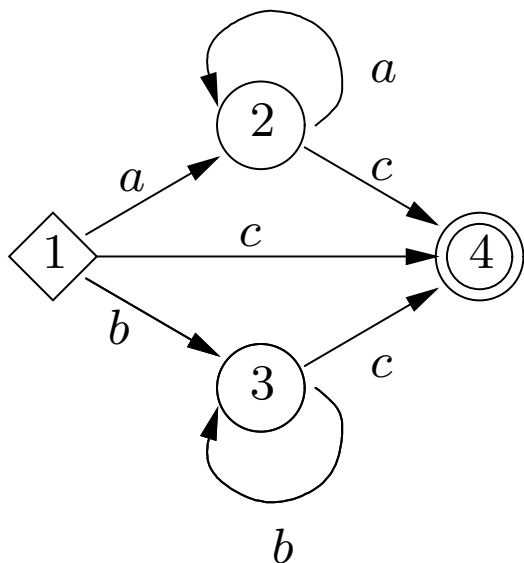
$$L_{1 \rightarrow 4}^4 = L_{1 \rightarrow 4}^2 \cup (L_{1 \rightarrow 3}^2 \cdot \{b, \epsilon\}^* \cdot \{c\})$$

and we apply the general formula further:

$$L_{1 \rightarrow 3}^2 = L_{1 \rightarrow 3}^1 \cup (L_{1 \rightarrow 2}^1 \cdot L_{2 \rightarrow 2}^1{}^* \cdot L_{2 \rightarrow 3}^1)$$

$$L_{1 \rightarrow 4}^2 = L_{1 \rightarrow 4}^1 \cup (L_{1 \rightarrow 2}^1 \cdot L_{2 \rightarrow 2}^1{}^* \cdot L_{2 \rightarrow 4}^1)$$

An example (continued)



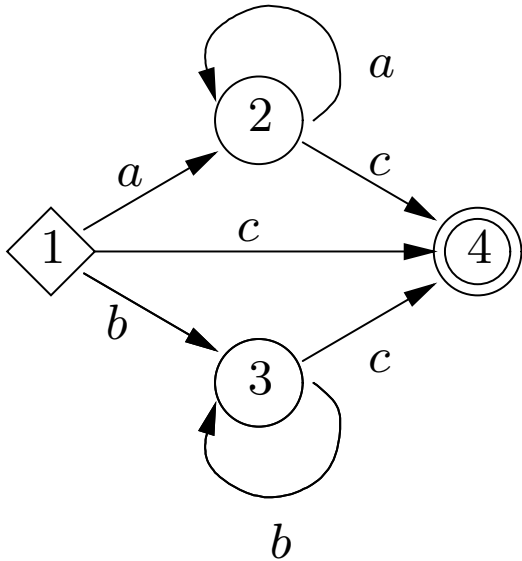
$$L_{1 \rightarrow 2}^1 = \{a\},$$

$$L_{2 \rightarrow 2}^1 = \{a, \epsilon\},$$

$$L_{1 \rightarrow 3}^1 = \{b\}, \quad L_{2 \rightarrow 3}^1 = \emptyset,$$

$$\text{and } L_{2 \rightarrow 4}^1 = \{c\}. \quad \text{So}$$

An example (continued)



$$L_{1 \rightarrow 2}^1 = \{a\},$$

$$L_{2 \rightarrow 2}^1 = \{a, \epsilon\},$$

$$L_{1 \rightarrow 3}^1 = \{b\}, \quad L_{2 \rightarrow 3}^1 = \emptyset,$$

$$\text{and } L_{2 \rightarrow 4}^1 = \{c\}. \quad \text{So}$$

$$L_{1 \rightarrow 3}^2 = \{b\} \cup (\{a\} \cdot \{a\}^* \cdot \emptyset) = \{b\}$$

$$L_{1 \rightarrow 4}^2 = \{c\} \cup (\{a\} \cdot \{a\}^* \cdot \{c\})$$

$$= \{a\}^* \{c\}$$

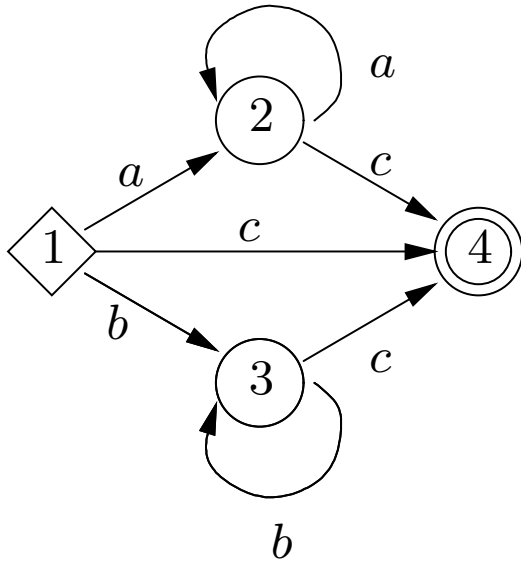
An example (continued)

$$\text{As } L_{1 \rightarrow 3}^2 = \{b\}$$

$$\text{and } L_{1 \rightarrow 4}^2 = \{a\}^* \{c\}$$

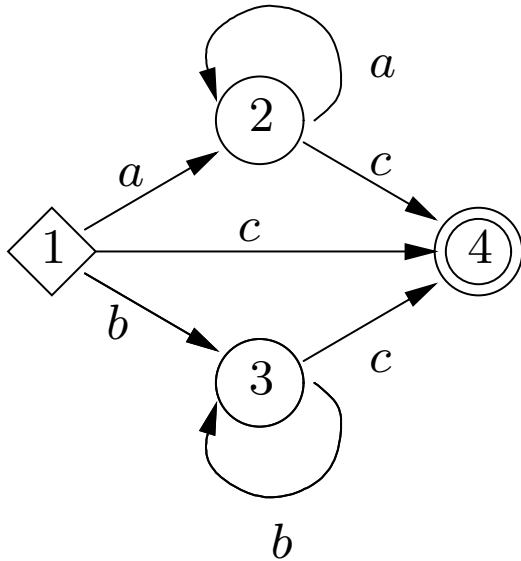
$$\begin{aligned} L_{1 \rightarrow 4}^4 &= L_{1 \rightarrow 4}^2 \cup (L_{1 \rightarrow 3}^2 \cdot \{b\}^* \cdot \{c\}) \\ &= \{a\}^* \{c\} \cup (\{b\} \cdot \{b\}^* \cdot \{c\}) \\ &= L(a * c | bb * c) \text{ is regular} \end{aligned}$$

An example (conclusion)



The language defined
(recognised) by this automaton is the
regular language $L(a^*c|bb^*c)$,

An example (conclusion)



The language defined
(recognised) by this automaton is the
regular language $L(a*c|bb*c)$,
which is the same language as
 $L(aa*c|c|bb*c)$.

Result

Summarizing all our efforts we have demonstrated the following result.

Theorem 1.5 *A language is regular if and only if it is recognized by a deterministic finite automaton, and if and only if it is recognized by a non-deterministic automaton.*

Result

In other words, we can use patterns for precisely the same languages as DFAs which we can use precisely for the same languages as NFAs.