A Subset of the URM Language; FA and NFA

0.0.1 Definition. If M is a FA, then its $L(M)$ is called the **regular set** associated with M, or even the **regular language** *recognised* (**decided**, actually) by M. □

This Note continues from where Note #9 left but we will present first a few more simple examples of automata* that decide /recognise some given set of strings over some alphabet.

0.0.2 Example. We want to specify (program!) an automaton M over $\Sigma = \{0, 1\}$, such that $L(M) = \{0^n 1 : n \geq 0\}$.

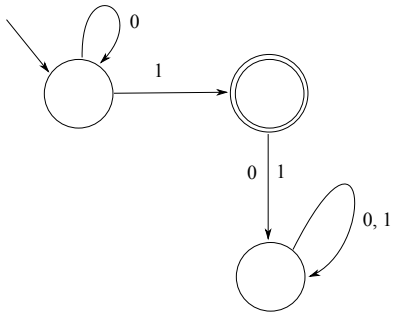
We recall that, for any string x, $x^0 =^{Def} \lambda$, while

$$x^{n+1} =^{Def} x^n * x =^{\text{induction!}} \overbrace{x * x * \ldots * x}^{n+1 \text{ copies of } x}$$

where I denoted concatenation by $*$. Thus the strings in $\{0^n 1 : n \geq 0\}$ are

$$1, 01, 001, 0001, 00001, \ldots \tag{1}$$

We readily see that the following automaton's **only accepting paths** will follow zero or more times the "loop" labeled 0 (attached to the start state), and then the edge labeled 1 to end up with an accepting state. Thus, its recognised language is indeed the one depicted intuitively in (1) above, that is, the set $\{0^n 1 : n \geq 0\}$.

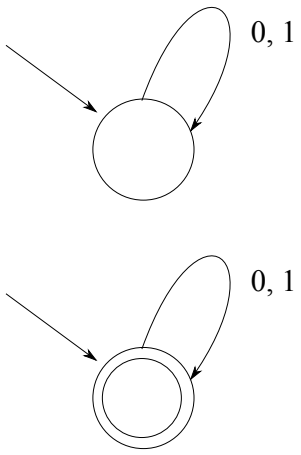


*Plural of automaton.

A new thing we learnt in this example is that in depicting an automaton as a graph we do *not* necessarily need to *name* the states! As in all mathematical arguments, we will of course assign names to objects (in particular to states) if we need to refer to them in the course of the argument —it is convenient to refer to them by name!

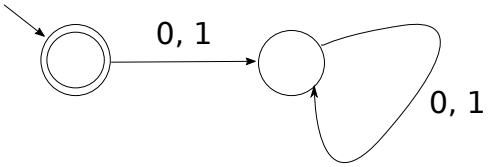
The reader should note the use of two shorthand notations in labeling: One, we used two labels on the vertical down-pointing edge. This abbreviates the use of *two* edges going from the accepting to the trap state, one labeled 0, the other 1. We could also have used the label "0, 1" at the left or right of the arrow, "," serving as a separator. This latter notational convention was used in labeling the loop attached to the trap state. □

0.0.3 Example. The two FAs below, each over the input alphabet $\{0, 1\}$, accept the languages $\emptyset$ (the top one) and $\{0, 1\}^*$ (the bottom one).



□

0.0.4 Example. The FA below over the input alphabet $\{0, 1\}$ accepts the language $\{\lambda\}$.



Indeed, we saw in Notes #9 that making the start (initial) state also accepting we do accept λ. Moreover, the FA above accepts nothing else since any input symbol leads to the rejecting *trap* state. □

0.1. Some Closure Properties of Regular Languages

0.1.1 Theorem. *The set of all regular languages over an alphabet Σ is closed under complement. That is, if $L \subseteq \Sigma^*$ is regular, then so is $\overline{L} =^{Def} \Sigma^* - L$.*

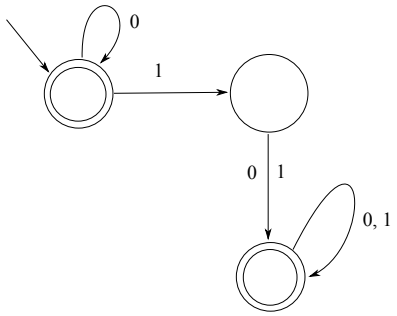
Proof. Let $L = L(M)$ for some FA M over input alphabet Σ and state alphabet Q. Moreover, let $F \subseteq Q$ be the set of accepting states of M.

We need a FA that recognises/decides $\overline{L}$.

Trivially, we want to swap the "yes" (accepting state) and "no" (rejecting state) behaviour of M, *changing nothing else*. Thus, $\overline{L} = L(\widetilde{M})$, where the FA $\widetilde{M}$ is the same as M, except that $\widetilde{M}$'s set of accepting states is $Q - F$. □

What makes the above proof tick is that FA are "**total**": Every input string will be scanned to its *eof*. Only the yes/no decision changes.

0.1.2 Example. The automaton that accepts the complement of the language in Example 0.0.2 is found without comment below, just following the construction of the $L(M)$ complement for some FA M, given above.



□

0.1.3 Theorem. *The set of all regular languages over an alphabet Σ is closed under union. That is, if $L \subseteq \Sigma^*$ and $L' \subseteq \Sigma^*$ are regular, then so is $L \cup L'$.*

Proof. So let $L = L(M)$ and $L' = L(M')$. The idea is to run M and M' in parallel on each input x and accept iff at least one of M and M' accepts.

We cannot use CT for this, as CT promises a full URM that decides the union. We want a FA though! So we will have to carefully simulate this parallelism with ONE FA! We must build one.

But consider: If indeed we had two separate FA, M and M' operate on input x simultaneously, then they would *at each step* **both** *consider the same input*

symbol, let us call it generically, a. However while M would be at state, say, q, M' would be at state, say, p. Let us say the response of M would be to get to state $\overline{q}$, i.e.,

$$M \text{ performs the move } (q) \xrightarrow{a} (\overline{q}) \tag{1}$$

while M' got into $\overline{q}'$, i.e.,

$$M' \text{ performs the move } (q') \xrightarrow{a} (\overline{q}') \tag{2}$$

IDEA! The "parallel" FA, let us call it N, will have the same alphabet Σ as M and M' and its states would be **ordered pairs** of states of M and M'. That is, if the we denote $Q = \{q_0, q_1, \ldots, q_n\}$ and $Q' = \{p_0, p_1, \ldots, p_m\}$ —where $n \neq m$ in general— the sets of states of M and M' respectively, then

$$\text{the set of states of } N \text{ is the set } Q \times Q' = \Big\{(q,p) : q \in Q \wedge p \in Q'\Big\}$$

Thus N keeps track of both computations —of M and M'— on input x by keeping track of the sequence of M-states caused by x using the first component of the state pair (q,p), while it keeps track of the sequence of M'-states caused by x using the second component of the state pair (q,p).

To do so, we build N so that a transition

$$(q,q') \xrightarrow{a} (\overline{q},\overline{q}') \tag{3}$$

is in N iff each of (1) and (2) above are in M and M' respectively. The accepting states for N will be the pairs (q,p) such that **at least one** of the components of the pair is accepting.

We omit a tedious induction proof that this N works as stated since the concept is straightforward. Once we introduce the Nondeterministic FA (NFA) a proof using NFA will be trivial. □

0.1.4 Corollary. *The set of all regular languages over an alphabet Σ is closed under intersection. That is, if $L \subseteq \Sigma^*$ and $L' \subseteq \Sigma^*$ are regular, then so is $L \cap L'$.*

Proof. Either by using 0.1.3 and 0.1.1 and noting that $L \cap L' = \overline{\overline{L} \cup \overline{L'}}$, or by the construction in 0.1.3, changing the accepting set of states in the FA N: The accepting states for N will be the pairs (q,p) such that **both** of the components of the pair is accepting. □

0.2. Proving Negative Results for FA; Pumping Lemma

Is there a FA M such that $L(M) = \{0^n 1^n : n \geq 0\}$? How can we tell? Surely, not by trying each FA (*infinitely many*) out there as a possible fit for this language!

The following theorem, known as the *pumping lemma* can be used to prove "negative" results such as: There is no FA M such as $L(M) = \{0^n 1^n : n \geq 0\}$. For short, the language $\{0^n 1^n : n \geq 0\}$ is **not regular**.

0.2.1 Theorem. (Pumping Lemma) *If the language S is regular, i.e., $S = L(M)$ for some FA M, then there is a constant C that we will refer to as* a pumping constant *such that for any string $x \in S$, if $|x| \geq C$, then we can decompose it as $x = uvw$ so that*

(1) $v \neq \lambda$

(2) $uv^i w \in S$, *for all* $i \geq 0$

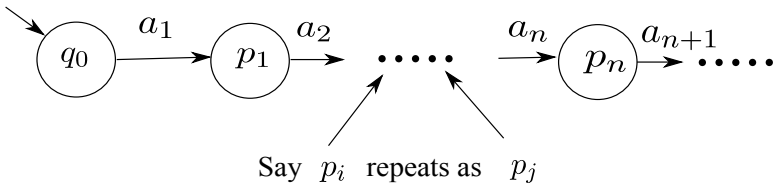
and

(3) $|uv| \leq C$.

A pumping constant is *not* uniquely determined by S.

Proof. So, let $S = L(M)$ for some FA M of n states. We will show that if we take $C = n$[†] this will work.

Let then $x = a_1 a_2 \cdots a_n \cdots a_m$ be a string of S. As chosen, it satisfies $|x| \geq C$. An accepting computation path of M with input x looks like this:



where $p_1, p_2, \ldots$ denotes a (notationally) convenient renaming[‡] of the states visited after q_0 in the computation. In the sequence

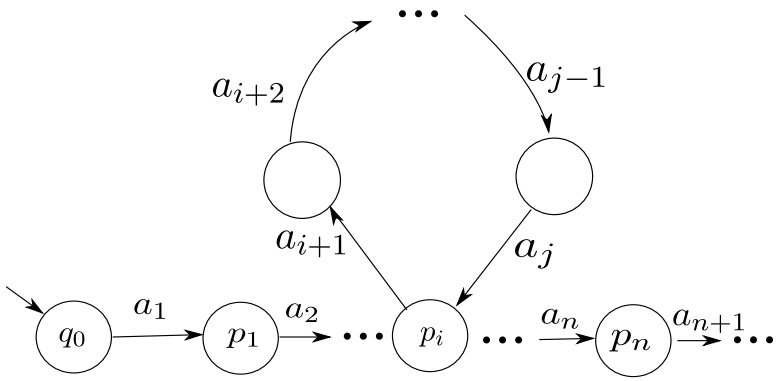
$$q_0, p_1, p_2, \ldots, p_n$$

we have named $n+1$ states, while we only have n states in the FA's "Q". Thus, at least two names refer to the same state —as we say, *two states repeat.*

[†]You see why C is not unique, since for any S that is an $L(M)$ we can have infinitely many different M that accept S. Can we not?

[‡]Why rename? What is wrong with $q_1, q_2, \ldots$? Well, the set Q is *given* as something like $\{q_0, q_1, q_2, q_3, \ldots\}$ using some arbitrary fixed enumeration order without repetition for its members. Now, it would be wrong to expect that the arbitrary input x caused the FA to walk precisely along q_1, q_2, q_3, etc., *after* it saw the first symbol of x. For one thing, it might need to go back to q_0 after it processed the first symbol. How do we show that after q_0 the states depend on input and they are neither distinct *necessarily* nor *necessarily* **the** $q_1, q_2, q_3, \ldots$? A messy way to do so is by double subscripting: $q_0, q_{j_1}, q_{j_2}, q_{j_3}, \ldots$ A less messy one is as I did above, using p_i.

We may redraw the computation above as follows:



We can now partition x into u, v and w parts from the picture above: We set

$$u = a_1 a_2 \dots a_i$$

$$v = a_{i+1} a_{i+2} \dots a_j$$

and

$$w = a_{j+1} a_{j+2} \dots a_m$$

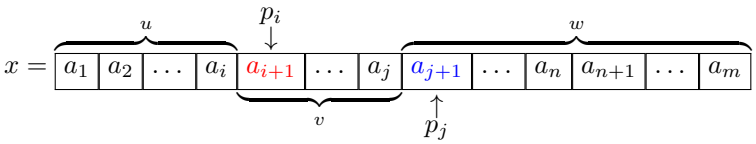
Note that

(1) $v \neq \lambda$, since there is at least one edge (labelled a_{i+1}) emanating from p_i on the sub-path that connects this state to the (identical) state p_j.

(2) We may utilize the loop v zero or more times (along with u in the front and w at the tail) to always obtain an accepting path. Thus, all of $uv^i w$ belong to $L(M)$ —i.e., S.

(3) Since $|uv| = j \leq n$, we have also verified that $|uv| \leq C$. □

The repeating pair p_i, p_j may occur anywhere between q_0 and p_n. A different graphical proof is common in the literature. Let $x = a_1 a_2 \dots a_m$ as above



Observe:

1. By determinism, the subcomputation that starts at symbol a_{j+1} (blue) —while in state p_j ($= p_i$)— will end at $q \in F$ at *eof* after consuming the string w.

2. By determinism, the subcomputation that starts at symbol a_{i+1} (red) in state p_i, will consume v and end at p_j, ready to process a_{j+1} (blue).

Thus, all of $uv, uvvw, uvvvw, \dots, uv^n w, \dots$ are in $L(M)$. Of course, so is $x = uvw$ (given).

0.2.2 Example. The language over $\{0,1\}$ given as $L = \{0^n1^n : n \geq 0\}$ is not regular. Suppose it is. Then the pumping lemma holds for L, so let C be an appropriate pumping constant and consider the string $x = 0^C1^C$ of L. We can then decompose x as uvw with $|uv| \leq C$ so that we can "pump" $v \neq \lambda$ as much as we like and the obtained uv^iw will all be in L.

We will prove the statement in red false, so we cannot pump; but then L cannot be regular![§]

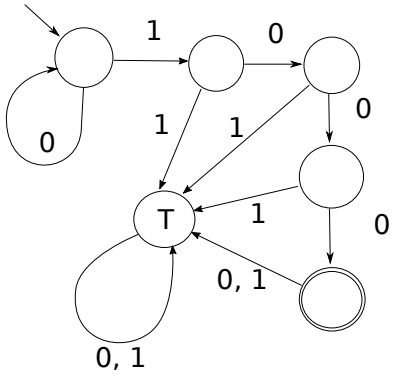
Red is false due to the observations: By $|uv| \leq C$, uv (and hence v) lie entirely in the 0^C-part of the chosen $x = 0^C1^C$. So, if we *pump down* (i.e., use v^0) we obtain $uw \in L$.

But $uw = 0^K1^C$ where $K < C$ since $|v| \geq 1$. But such unbalanced 0-1 string cannot be in L, by specification, so we contradicted the pumping lemma. □

All proofs by 0.2.1 are by contradiction and they prove *non recognisability* by *any* FA (or, equivalently, NFA to be introduced in Section 0.3).

0.2.3 Example. We introduced FA as special URMs that cannot write. Is it immediate that they cannot compute functions? No, for we can agree that by "compute $f(x)$" we mean "decide the graph $y = f(x)$".

For example, we can "compute" $\lambda x.3$ by accepting all strings, but no others, of the form 0^n1000 over the alphabet $\{0,1\}$. That is we use 1 as a separator between input n (depicted as 0^n) and output 3 (depicted as 000), then the following FA decides (recognises) the language $L = \{0^n1000 : n \geq 0\}$.



□

0.2.4 Example. FA cannot compute $\lambda x.x + 1$. "Surely", you say, "how can they add 1 if they cannot do arithmetic or write anything at all?"

Again, how about deciding the "graph"-language over $A = \{0,1\}$, given by $T = \{0^n10^{n+1} : n \geq 0\}$? Here "$0^n$" represents input n, "0^{n+1}" represents output $n+1$ and 1 is a separator as in the previous example.

[§] All sufficiently long strings of regular languages can be pumped by 0.2.1 and stay in L.

Alas, no FA can do this. Say T is FA-decidable, and let C be an appropriate pumping constant. Choose $x = 0^C 10^{C+1}$. Splitting x as uvw with $|uv| \leq C$ we see that 1 is to the right of v. Thus, uw (using v^0) is not in T since the relation between the 0s to the left and those to the right of 1 is destroyed. This contradicts the assumption that T is FA-decidable. □

0.2.5 Exercise. Indeed FA cannot even compute the identity function, $\lambda x.x$, as it should be clear from 0.2.2. Adapt that proof to show the graph language for $\lambda x.x$, namely, $\{0^n 10^n : n \geq 0\}$ is not regular. □

0.2.6 Example. The set over the alphabet $\{0\}$ given by $P = \{0^q : q \text{ is a prime number}\}$ is not FA-decidable.

Assume the contrary, and let C be an appropriate pumping constant. Let $Q \geq C$ be prime. We show that considering the string $x = 0^Q$ will lead us to a contradiction. Well, as x is longer than C, let us write —according to 0.2.1— $x = uvw$. By said theorem, we must have that all numbers $|u| + i|v| + |w|$, for $i \geq 0$ are prime. These numbers have the form

$$ai + b \tag{1}$$

where $a = |v| \geq 1$ and $b = |u| + |w|$. Can all these numbers in (1) (for all i) be prime?

Here is why not, and hence our contradiction. We consider cases:

- Case where $b = 0$. This is impossible, since the numbers in (1) now have the form ai. But, e.g., $a4$ is not prime.
- Case where $b > 0$. Now taking $i = b$, one of the numbers of the form (1) is $(a+1)b$. We have Subcases!
  - Subcase $b > 1$. Then $(a+1)b$ is not prime (recall that $a+1 \geq 2$ since $a \geq 1$).
  - Subcase $b = 1$. Then take $i = 2 + a$ to obtain the number (of type (1)) $a(2+a)+1 = a^2+2a+1 = (a+1)^2$. But this is not prime! □

The preceding shows that we can have a set that is sufficiently complex and thus fails to be FA-decidable *even over a single-symbol alphabet*. Here is another such case.

0.2.7 Example. Consider $Q = \{0^{n^2} : n \geq 0\}$ over the alphabet $A = \{0\}$. It will not come as a surprise that Q is not FA-decidable.

For suppose it is. Then, if C is an appropriate pumping constant, consider $x = 0^{C^2}$. Clearly, $x \in Q$ and is long enough. So, split it as $x = uvw$ with $|uv| \leq C$ and $v \neq \lambda$.

Now, by 0.2.1,

$$uvvw \in Q \tag{1}$$

But

$$C^2 \leq |uvvw| \leq |uvw| + |uv| \leq C^2 + C < C^2 + 2C + 1 = (C+1)^2$$

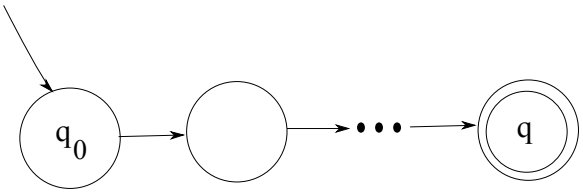
Thus, the number $|uvvw|$ is not a perfect square being between two successive ones. But this will not do, because by (1), for some n, we have $uvvw = 0^{n^2}$ and thus $|uvvw| = n^2$ —a perfect square after all! □

0.3. Nondeterministic Finite Automata

The FA formalism provides us with tools to finitely define certain languages: Such a language —defined as an $L(M)$ over some alphabet A, for some FA M— contains a string x iff there is an *accepting computation*

$$q_0 x \vdash_M^* xq \tag{1}$$

or, equivalently an *accepting path* within the FA (given as a flow diagram) whose labels from left to right form x.

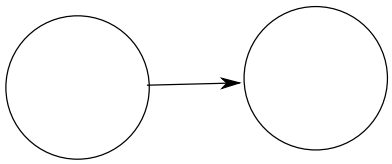


The computation (1) above, equivalently, the path labeled x within the FA, are uniquely determined by x since the automaton's δ —the *transition relation*— is a total *function*.

Much is to be gained in theoretical flexibility if we **relax both** the requirements that δ is single-valued (a function) **and** total.

This gives rise to a *nondeterministic* model of finite automata, an "NFA", that may accept a string in more than one ways, that is, there may be more than one distinct paths from q_0 to an accepting state q, each labeled by the same string x.

Indeed, even more flexibility is attained if we also allow "*unconditional jumps*" from one state to another, such as



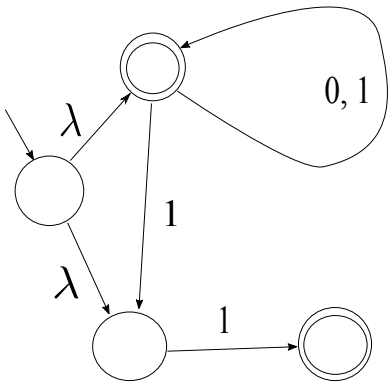
which, in the first approximation, will have unlabeled edges as above. However, in order to retain the central property that we may "read off" what string is

accepted by any given accepting path, namely, *by simply concatenating the edge labels of the path from left to right*, we will label all unconditional jumps by a string that is *neutral with respect to concatenation* —that is, by λ. For this reason, unconditional jumps are also called "λ moves".

0.3.1 Example. The displayed flow diagram below, over the alphabet $\{0, 1\}$, incorporates all the liberties in notation and convention introduced in the preceding discussion.

We have two λ moves, and the string 1 can be accepted in two distinct ways: One is to follow the top λ move, and then go once around the loop, consuming input 1. The other is to follow the bottom λ move, and then follow the transition labeled 1 to the accepting state at the bottom (reading 1 in the process).

Folklore jargon will have us speak of *guessing* when we describe what the diagram does with an input. For example, to accept the input 00 one would say that *the NFA guesses* that it should follow the upper λ, and then it would go twice around the top loop, on input 0 in each case.



This diagram is an example of a *nondeterministic finite automaton*, or NFA; it has λ moves, its transition relation —as depicted by the arrows— is not a function (e.g., the top accepting state has two distinct responses on input 1), nor is it total. For example, the bottom accepting state is not defined on any input; nor is the start state: λ is not an input! □

Returning to the issue of *guessing*, we emphasize that a NFA simply provides the *mathematical framework* within which *we* can formulate and verify an *existential statement* of the type

for some given input x, an accepting path exists

Given an acceptable input, the NFA does not actually guess "correct" moves (from among a set of choices), either in a hidden manner (consulting the Oracle in Delphi, for example), or in an explicit computational manner (e.g., parallelism, backtracking) toward *finding* an accepting path for said x. Simply, the

NFA formalism allows *us* to *state*, and provides tools so that we can *verify*, the statement

$$\textit{for some given input } x \textit{ an accepting path} \text{ exists } (0.3.3) \tag{1}$$

just as the language of logic allows us to *state* statements such as $(\exists y)\mathscr{F}(y,x)$, and offers tools for their proof.

An independent agent, which could be ourselves or a FA —yes, we will see that every NFA can be can simulated by some FA!— can effect the verification that indeed an accepting path labeled x exists.

0.3.2 Definition. (NFA as flow diagrams) A *nondeterministic finite automaton*, or *NFA*, over the *input alphabet* Σ is a finite directed graph of circular nodes —the *states*— and interconnecting edges —the *transitions*— the latter labeled by members of A. It is specified as in the definition of FA (Notes #9) with some amendments:

- The restriction (for FA) that for every state L and every $a \in \Sigma$, there will be precisely *one* edge, labeled a, leaving L and pointing to some state M (possibly, $L = M$) is removed.
- The NFA *need not be fully specified* —it can be *nontotal*.
- It is allowed unconditional jumps, that is, edges labeled only by λ.

We will generically use names such as M, N, or K for NFA, just as we did for the case of FA. □

The above can be recast in an algebraic formulation, viewing a NFA from an alternative point of view as a tuple of ingredients Q, A, etc., just as we did for the FA. If one does so, one will relax the requirement that δ be a single-valued relation, and will also relax the requirement of totalness. One will also allow δ to have inputs of the type $\langle q, \lambda\rangle$ making sure to view this *as an extension of what δ can deal with* (as inputs) rather than mistaking this as an extension of the input alphabet. Said alphabet cannot have the empty string as a member.

We will not pursue the algebraic model, as the flow diagram model will do all we want it to do.

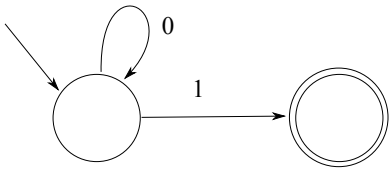
It is trivial that, since a NFA is defined by relaxing requirements in the Definition of the FA (Notes #9), any FA is also a NFA, but not conversely, as the preceding example demonstrates.

The NFA "computes" as follows:

0.3.3 Definition. (NFA computations; string acceptance) Let M be a NFA over the input alphabet Σ. An *accepting path* is a path in M from the start state to some accepting state.

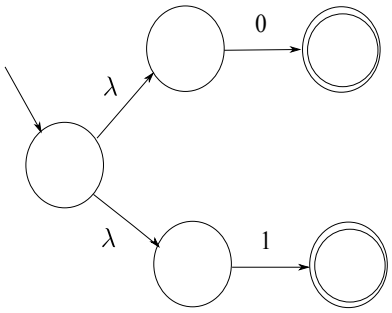
A string x over Σ is accepted by M iff x is obtained by concatenating the path labels from left to right in an accepting path. We say that x *names*, or is the name of, said accepting path. $L(M)$ denotes the set of all strings over Σ accepted by M. We say that M decides/recognises $L(M)$. □

0.3.4 Example. The following is a NFA but not a FA (why? Compare with 0.0.2). It decides the language $\{0^n1 : n \geq 0\}$.



□

0.3.5 Example. NFA are much easier to construct than FA, partly because of the convenience of the λ moves, and the lack of concern about single-valuedness of δ. Also, partly due to lack of concern for totalness: we do not have to worry about "installing" a trap state. For example, the following NFA over $A = \{0, 1\}$ decides/recognises just its alphabet A and nothing else as we can trivially see that there are just two accepting paths: one named "0" and one named "1".



□

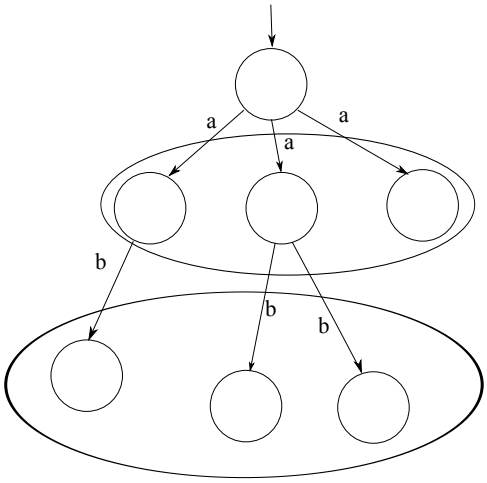
0.3.1. From FA to NFA and Back

We noted earlier that any FA is a NFA, thus the NFA are at least as powerful as the FA. They can do all that the deterministic model can do. It is a bit of a surprise that the opposite is also true: For every NFA M we can *construct* a FA N, such that $L(M) = L(N)$.

Thus, in the case of these very simple machines, nondeterminism ("guessing") buys convenience, but not real power.

How does one simulate a NFA on an input x? The most straightforward idea is to trace *all possible paths* labeled x (due to nondeterminism they may be more than one —or none at all) *in parallel* and accept iff one (or more) of

those is accepting. The principle of this idea is illustrated below.



Say, the input to the NFA M is $x = ab\ldots$ Suppose that a leads the start state —which is at "level 0"— to three states; we draw all three. These are at level 1. We repeat for each state at level 1 on input b: Say, for the sake of discussion, that, of the three states at level 1, the first leads to one state on input b, the second leads to two and the third leads to none. We draw these three states obtained on input b; they are at level 2. Etc.

An FA can keep track of all the states at the various levels since they can be no more than the totality of states of the NFA M! The amount of information at each level is independent of the input size —i.e., it is a constant— and moreover can be coded as a single FA-state (depicted in the figure by an ellipse) that uses a "compound" name, consisting of all the NFA state names at that level. This has led to the idea that the simulating FA must have as states nodes whose names are *sets of* state names of the NFA. Clearly, for this construction, state names are important, through which we can keep track of and describe what we are doing. Here are the details:

0.3.6 Definition. (a-successors) Let M be a NFA over an input alphabet Σ, q be a state, and $a \in \Sigma$. A state p is *an a-successor* of q iff there is an edge from q to p, labeled a. □

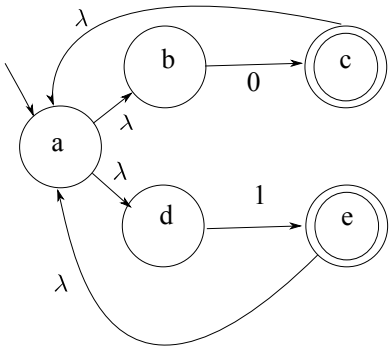
In a NFA a-successors need not be unique, nor need to exist (for every pair (q, a)). On the other hand, in a FA they exist and are unique.

0.3.7 Definition. (λ-closure) Let M be a NFA with state-set Q and let $S \subseteq Q$. The λ-closure of S, denoted by $\lambda(S)$, is defined to be the smallest set that includes S but also includes all $q \in Q$, such that there is a path, named λ, from some $p \in S$ to q.

When we speak of the λ-closure of a state q, we mean that of the set $\{q\}$ and write $\lambda(q)$ rather than $\lambda(\{q\})$. □

Note that a path named λ will have **all** its edges named λ since the concatenation of a sequence of strings is λ iff each string in the sequence is.

0.3.8 Example. Consider the NFA below.



We compute some λ-closures: $\lambda(a) = \{a, b, d\}$; $\lambda(c) = \{c, a, b, d\}$. □

0.3.9 Theorem. *Let M be a NFA with state set Q and input alphabet Σ. Then there is a FA N that has as state set a subset of $\mathcal{P}(Q)$ —the power set of Q— and the same input alphabet as that of M. N satisfies $L(M) = L(N)$.*

We say that two automata M and N (whether both are FA or both are NFA, or we have one of each kind) are *equivalent* iff $L(M) = L(N)$. Thus, the above says that for any NFA there is an equivalent FA. In fact, this can be strengthened as the proof shows: We can *construct* the equivalent FA.

Proof. The FA N will have as state set some subset of $\mathcal{P}(Q)$, meaning that every state of N will have a compound name consisting of the names (in any order, hence *set* rather than sequence) of the members of some subset of Q. Moreover,

- The start state of N is $\lambda(q_0)$, where q_0 is the start state of M.
- A state of N is accepting iff its *name* contains at least one accepting state *name* of M.
- Let S be a state of N and let $a \in \Sigma$. The unique a-successor of S in N is constructed as follows:

  (1) Construct the set of *all* a-successors in M of *all* members of S. Call T this set of a-successors.

  (2) Construct $\lambda(T)$; this is *the* a-successor of S in N.

As an illustration, we compute some *0-successors* in the FA constructed as above if the given NFA is that of Example 0.3.8.

(I) For state $\{a, b, d\}$ step (1) yields $\{c\}$. Step (2) yields the λ-closure of $\{c\}$: The state $\{c, a, b, d\}$ is the 0-successor.

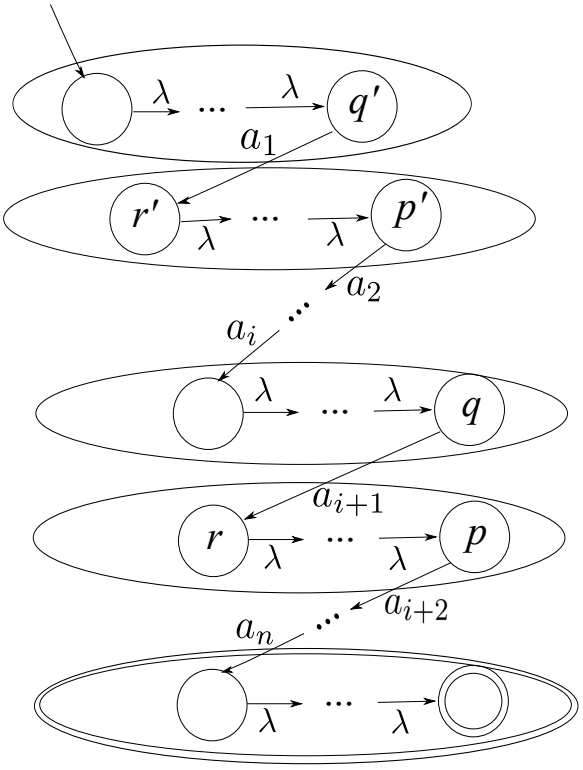
(II) For state $\{c, a, b, d\}$ step (1) yields $\{c\}$. Step (2) yields the λ-closure of $\{c\}$: The state $\{c, a, b, d\}$ is the 0-successor; that is, the 0-edge loops back to where it started.

We return to the proof. We will show that this construction of the FA N works. So let us prove first that $L(M) \subseteq L(N)$.

Let $x = a_1 a_2 \cdots a_n \in L(M)$. Without loss of generality, we have an accepting path in M that is labeled as follows:

$$\lambda^{j_1} a_1 \lambda^{j_2} a_2 \lambda^{j_3} a_3 \cdots \lambda^{j_n} a_n \lambda^{j_{n+1}} \tag{3}$$

where each λ^{j_i} depicts $j_i \geq 0$ consecutive path edges, each labeled λ, *where $j_i = 0$ in this context means that the j_i group has no λ-moves.* We show this M-path graphically below as the zig-zag path with horizontal segments alternating with down-sloping segments. For easy reference, we assign the *level* number i to each horizontal part labeled $\lambda^{j_{i+1}}$, that is the one that is followed immediately by a down-sloping edge labeled a_{i+1}.



Now the FA N that is constructed as detailed in the above three bullets will —its δ being total— have *a unique*[¶] *path* labeled $a_1 a_2 \ldots a_n$ from its start state to some other state, its states being denoted by ellipses in the diagram.

[¶]By determinism.

The diagram *implicitly claims* that each N-state at level i, depicted as an ellipse in the diagram,

(A) contains *as part of its name* all the states that participate in the M-subpath with name $\lambda^{j_{i+1}}$, for $i = 0, 1, 2, \ldots$

(B) the N-state at level i has the N-state shown, and *partially named*, at the next level as its a_{i+1}-successor.

We prove (A) by an easy induction on i and obtain (B) at once (bullet three on p.14) as a side effect. Indeed, for level $i = 0$ (basis), all the state names shown (from M) are in $\lambda(q_0)$ and we are done by the first bullet on p.14.

Taking as I.H. the validity of (A) for a fixed unspecified level i —where we have an N-state S— we look next at level $i + 1$, where we have an N-state T.

By definition of "level", level $i+1$ is reached by the edge named a_{i+1}. By the assumption on how M accepts x, the edge named a_{i+1} bridges the two indicated states, q and r. From what we know about the components present in the name of S (I.H.), the third bullet that describes the transitions of N entails that all states in the λ-path from r to p are in the N-state T.

The string will be accepted by N iff the state pointed at by a_n —at level n— is *accepting in* N. This is so by the fact that the last state of the M-computation is accepting and —by (A)— is part of the last N-state in the above N-path.

We turn to the converse. So let $x = a_1 a_2 \ldots a_n \in L(N)$. We will argue that $x \in L(M)$.

We will reuse the above figure. Let us concentrate at first only in the elliptical states of the FA N and the indicated in the figure interconnecting transitions, which from top to bottom form the accepting N-computation path labeled $a_1 a_2 \ldots a_n$. We will want to produce a corresponding accepting M-path, that we will "fold" and fit its "horizontal" (λ-moves) parts inside appropriate ellipse states. This time it will be most convenient to do so starting at the bottom of the FA path and work backwards.

Thus the last (bottommost) state of N is two things:

(i) *accepting*; hence must contain an accepting M-state (as illustrated)

(ii) *the a_n-successor* of the preceding N-state (not illustrated). Therefore (p.14), the latter *must* contain an M-state from which the a_n-edge emanates and this edge either points to the illustrated accepting M-state, or, *more generally*, to a different M-state, which is part of the last ellipse's name *and* is connected to the accepting M-state by an λ-path. *We always adopt the general case* (illustrated).

Continuing to build an M-computation path backwards, from an accepting state toward the start state —and "folding it" inside the ellipses of N as we go, leaving only the edges labeled a_j to connect levels— assume that we have reached the ellipse at level $i+1$. This N-state acts on input a_{i+2} that emanates from the M-state p inside the ellipse. By the third bullet on p.14, this p must have become

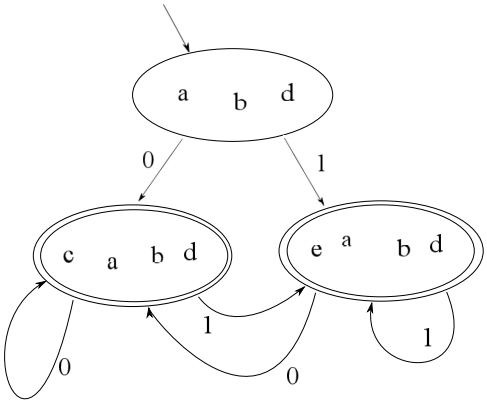
part of the ellipse's name either by directly being pointed to by the a_{i+1}-edge —emanating from some M-state, that we will call q, inside the i-level ellipse— or the edge from q actually points to a state r inside the level-$i+1$ ellipse, this r, in turn, pointing to p via an λ-path (the "general case"), as illustrated.

Once we reach level 1 in this way, we have a state p' (of M) as part of the level-1 ellipse's name, which —as in the general case at level $i+1$— is either pointed to by the a_1-edge (emanating from q') directly, or indirectly via an λ-path from r'; as illustrated. Finally, at level 0, we have N's start state. Thus, either q' is the start state of M, or, more generally, as illustrated, q' is reachable from M's start state by an λ path. We have constructed an accepting path in M, labeled x. □

In theory, to construct a FA for a given NFA we draw all the states of the latter and then determine the interconnections via edges, for each state-pair and each member of the input alphabet Σ. In practice we may achieve significant economy of effort if we start building the FA "from the start state down": That is, starting with the start state (level 0) we determine *all* its a-successors, for each $a \in \Sigma$. At the end of this step we will have drawn all states at "level 1". In the next step for each state at level 1, draw its a-successors, for each $a \in \Sigma$. And so on.

This sequence of steps terminates since there are only a finite number of states in the FA and we cannot keep writing *new* ones; that is, sooner or later we will stop introducing new states: edges will point "back" to existing states. See the following example.

0.3.10 Example. We convert the NFA of 0.3.8 to a FA. See below, and review the above comment and the proof of 0.3.9, in particular the three bullets on p.14, to verify that the given is correct, and follows procedure.



You will notice the aforementioned economy of effort achieved by our process. We have only three states in the FA as opposed to the predicted 32 ($= 2^5$) of the proof of Theorem 0.3.9. But what happened to the other states? Why are they not listed by our procedure?

Because the procedure only constructs FA states that *are accessible by the start state via a computation path.* These are the only ones that can possibly participate in an accepting path. The others are irrelevant to accepting computations —indeed to any computations that start with the start state— and can be omitted without affecting the set decided by the FA. □

0.3.11 Example. Suppose that we have converted a NFA M into a FA N. Let a be in the input alphabet. What is the a-successor of the state named $\emptyset$ *in* N? Well, there is no M state q that connects some state in $\emptyset$ to q with label a. Thus, the set of a-successors (*in* M) of states from $\emptyset$ is itself the empty set. In other words, the a-successor of $\emptyset$ *in* N is $\emptyset$. The edge labeled a loops back to it.

Therefore, in the context of the NFA-to-FA conversion, $\emptyset$ is a *trap state* in N.

□