Automata & languages
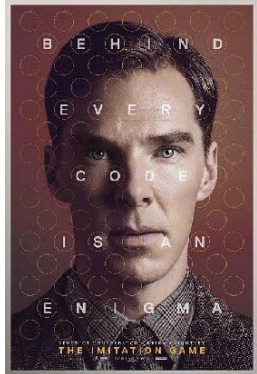
A primer on the Theory of Computation

Laurent Vanbever

www.vanbever.eu

ETH Zürich (D-ITET)

September, 26 2019

Last week was all about

Deterministic Finite Automaton

We saw three main concepts

Regular Language

Formal definition

Closure

Regular Language

A language *L* is *regular* if some finite automaton recognizes it
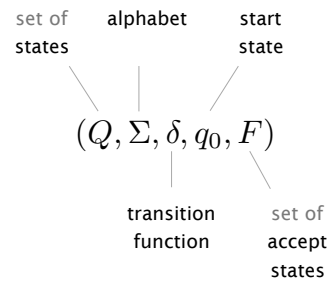
Formal definition

Closure

Regular Language

Formal definition

A finite automaton is a 5-tuple

$$(Q, \Sigma, \delta, q_0, F)$$

Closure

set of states   alphabet   start state

$$(Q, \Sigma, \delta, q_0, F)$$

transition function   set of accept states

Regular Language

Formal definition

Closure

If L_1 and L_2 are regular, then so are:

$L_1 \cup L_2$ $L_1 \cap L_2$ $\overline{L_1}$

$L_1 \oplus L_2$ $L_1 - L_2$

Back to Nondeterministic FA

- Question: Draw an FA which accepts the language

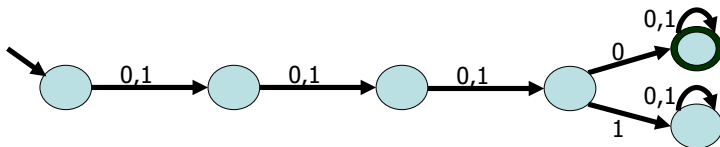 $L_1 = \{ x \in \{0,1\}^* \mid 4^{th}$ bit from left of x is 0 $\}$

- FA for L_1:
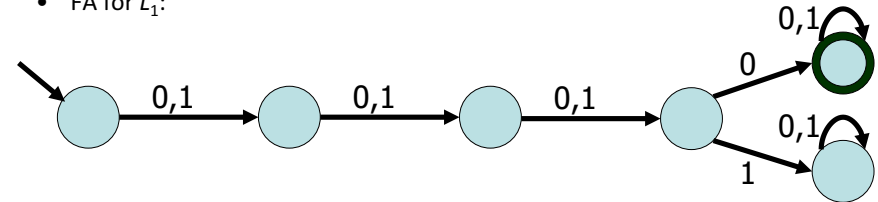
- Question: What about the 4th bit from the *right*?

- Looks as complicated: $L_2 = \{ x \in \{0,1\}^* \mid 4^{th}$ bit from right of x is 0 $\}$
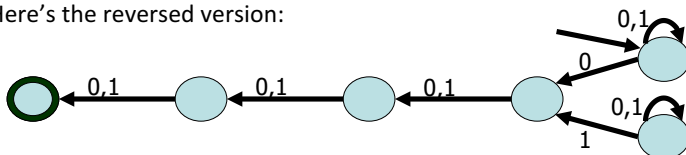

Weird Idea

- Notice that L_2 is the reverse L_1.

- I.e. saying that 0 should be the 4th from the left is reverse of saying that 0 should be 4th from the right. Can we simply reverse the picture (reverse arrows, swap start and accept)?!?

- Here's the reversed version:

Discussion of weird idea

1. Silly unreachable state. *Not pretty, but allowed in model.*

2. Old start state became a crashing accept state. *Underdeterminism. Could fix with fail state.*

3. Old accept state became a state from which we don't know what to do when reading 0. *Overdeterminism. Trouble.*

4. (Not in this example, but) There could be more than one start state! *Seemingly outside standard deterministic model.*

- Still, there is something about our automaton. It turns out that NFA's (=Nondeterministic FA) are actually quite useful and are embedded in many practical applications.

- Idea, keep more than 1 active state if necessary.

Introduction to Nondeterministic Finite Automata

- The static picture of an NFA is as a graph whose edges are labeled by Σ and by ε (together called Σ_ε) and with start vertex q_0 and accept states F.

- Example:
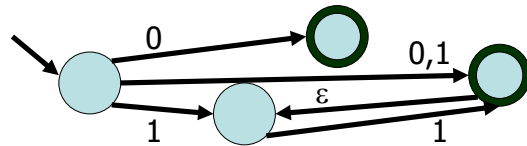
- Any labeled graph you can come up with is an NFA, as long as it only has *one start state*. Later, even this restriction will be dropped.

NFA: Formal Definition.

- Definition: A nondeterministic finite automaton (NFA) is encapsulated by $M = (Q, \Sigma, \delta, q_0, F)$ in the same way as an FA, except that δ has different domain and co-domain: $\delta : Q \times \Sigma_\varepsilon \rightarrow P(Q)$

- Here, $P(Q)$ is the power set of Q so that $\delta(q,a)$ is the set of all endpoints of edges from q which are labeled by a.
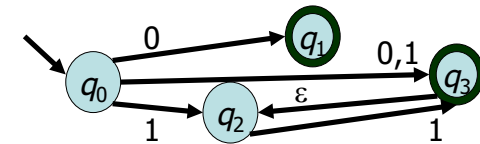
- Example, for NFA of the previous slide:

$\delta(q_0,0) = \{q_1,q_3\},$
$\delta(q_0,1) = \{q_2,q_3\},$
$\delta(q_0,\varepsilon) = \varnothing,$

...

$\delta(q_3,\varepsilon) = \{q_2\}.$

Formal Definition of an NFA: Dynamic

- Just as with FA's, there is an implicit auxiliary tape containing the input string which is operated on by the NFA. As opposed to FA's, NFA's are parallel machines – able to be in several states at any given instant. The NFA reads the tape from left to right with each new character causing the NFA to go into another set of states. When the string is completely read, the string is accepted depending on whether the NFA's final configuration contains an accept state.

- Definition: A string *u* is accepted by an NFA *M* iff there *exists a path* starting at q_0 which is labeled by *u* and ends in an accept state. The language accepted by M is the set of all strings which are accepted by *M* and is denoted by *L(M)*.
  - Following a label ε is for free (without reading an input symbol). In computing the label of a path, you should delete all ε's.
  - The only difference in acceptance for NFA's vs. FA's are the words "*there exists*". In FA's the path always exists and is unique.
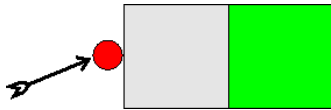
Example

M_4:
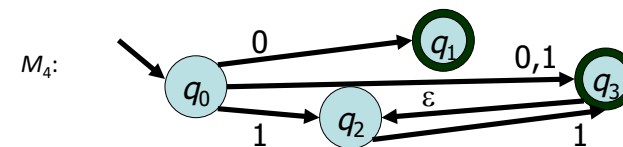
Question: Which of the following strings is accepted?
1. ε
2. 0
3. 1
4. 0111

NFA's vs. Regular Operations

- On the following few slides we will study how NFA's interact with regular operations.

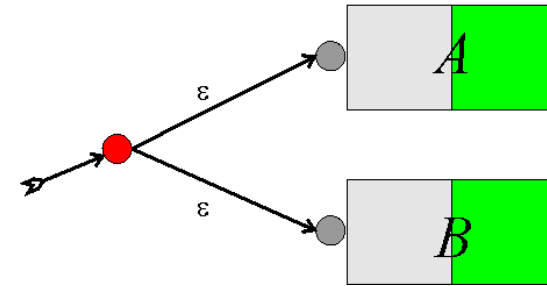- We will use the following schematic drawing for a general NFA.

- The red circle stands for the start state q_0, the green portion represents the accept states F, the other states are gray.
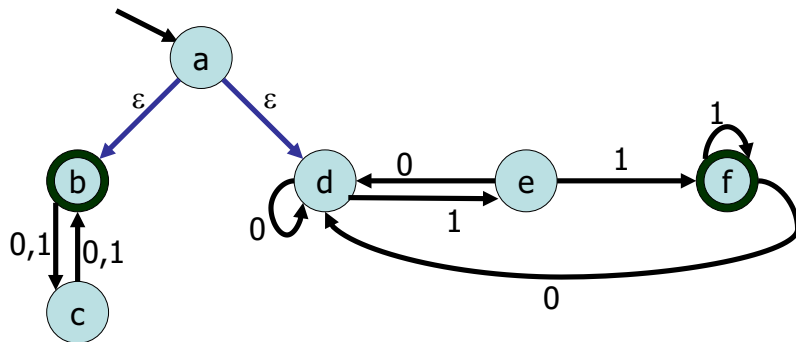
NFA: Union

- The union $A \cup B$ is formed by putting the automata A and B in parallel. Create a new start state and connect it to the former start states using ε-edges:
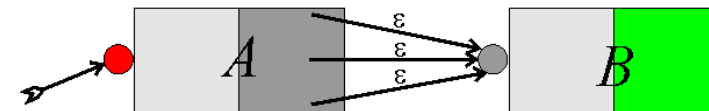
Union Example

- L = {x has even length} $\cup$ {x ends with 11}
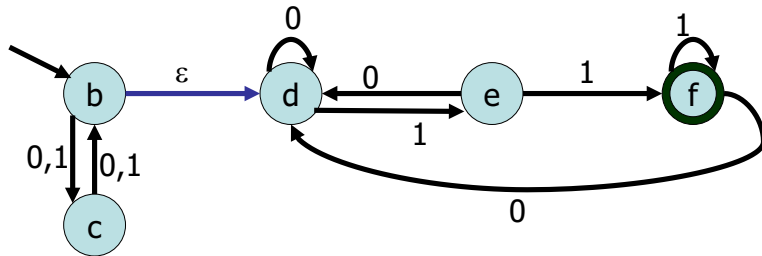
NFA: Concatenation

- The concatenation $A \bullet B$ is formed by putting the automata in serial. The start state comes from A while the accept states come from B. A's accept states are turned off and connected via ε-edges to B's start state:
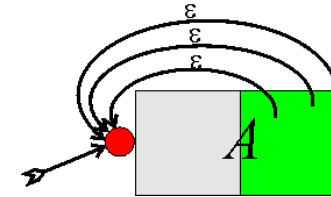
Concatenation Example

- $L = \{x \text{ has even length}\} \bullet \{x \text{ ends with } 11\}$

- Remark: This example is somewhat questionable…
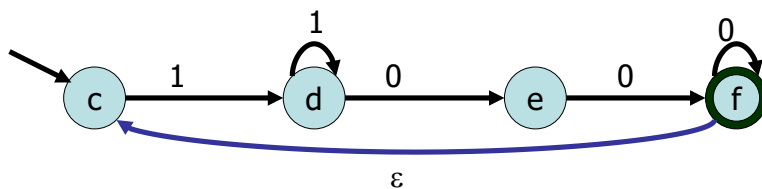
NFA's: Kleene-+.

- The Kleene-+ A^+ is formed by creating a feedback loop. The accept states connect to the start state via ε-edges:

Kleene-+ Example

$$L = \left\{ \begin{array}{l} x \text{ is a streak of one or more 1's followed} \\ \text{by a streak of two or more 0's} \end{array} \right\}^+$$

$$= \left\{ \begin{array}{l} x \text{ starts with 1, ends with 0, and alternates} \\ \text{between one or more consecutive 1's} \\ \text{and two or more consecutive 0's} \end{array} \right\}$$

NFA's: Kleene-*

- The construction follows from Kleene-+ construction using the fact that A^* is the union of A^+ with the empty string. Just create Kleene-+ and add a new start accept state connecting to old start state with an ε-edge:
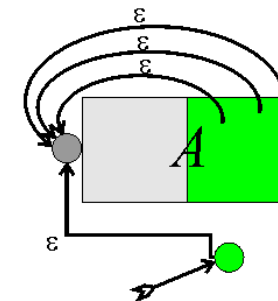
Closure of NFA under Regular Operations

- The constructions above all show that NFA's are *constructively* closed under the regular operations. More formally,

- Theorem: If L_1 and L_2 are accepted by NFA's, then so are $L_1 \cup L_2$, $L_1 \bullet L_2$, L_1^+ and L_1^*. In fact, the accepting NFA's can be constructed in linear time.

- This is almost what we want. If we can show that all NFA's can be converted into FA's this will show that FA's – and hence regular languages – are closed under the regular operations.

Regular Expressions (REX)

- We are already familiar with the regular operations. Regular expressions give a way of symbolizing a sequence of regular operations, and therefore a way of generating new languages from old.

- For example, to generate the regular language {banana,nab}* from the atomic languages {a},{b} and {n} we could do the following:

$$((\{b\} \bullet \{a\} \bullet \{n\} \bullet \{a\} \bullet \{n\} \bullet \{a\}) \cup (\{n\} \bullet \{a\} \bullet \{b\}))^*$$

Regular expressions specify the same in a more compact form:

$$(banana \cup nab)^*$$

Regular Expressions (REX)

- Definition: The set of regular expressions over an alphabet Σ and the languages in Σ^* which they generate are defined recursively:
  - Base Cases: Each symbol $a \in \Sigma$ as well as the symbols ε and $\varnothing$ are regular expressions:
    - a generates the atomic language $L(a) = \{a\}$
    - ε generates the language $L(\varepsilon) = \{\varepsilon\}$
    - $\varnothing$ generates the empty language $L(\varnothing) = \{ \} = \varnothing$
  - Inductive Cases: if r_1 and r_2 are regular expressions so are $r_1 \cup r_2$, $(r_1)(r_2)$, $(r_1)^*$ and $(r_1)^+$:
    - $L(r_1 \cup r_2) = L(r_1) \cup L(r_2)$, so $r_1 \cup r_2$ generates the union
    - $L((r_1)(r_2)) = L(r_1) \bullet L(r_2)$, so $(r_1)(r_2)$ is the concatenation
    - $L((r_1)^*) = L(r_1)^*$, so $(r_1)^*$ represents the Kleene-*
    - $L((r_1)^+) = L(r_1)^+$, so $(r_1)^+$ represents the Kleene-+

Regular Expressions: Table of Operations including UNIX

Operation	Notation	Language	UNIX
Union	$r_1 \cup r_2$	$L(r_1) \cup L(r_2)$	$r_1	r_2$
Concatenation	$(r_1)(r_2)$	$L(r_1) \bullet L(r_2)$	$(r_1)(r_2)$
Kleene-*	$(r)^*$	$L(r)^*$	$(r)^*$
Kleene-+	$(r)^+$	$L(r)^+$	$(r)^+$
Exponentiation	$(r)^n$	$L(r)^n$	$(r)\{n\}$

Regular Expressions: Simplifications

- Just as algebraic formulas can be simplified by using less parentheses when the order of operations is clear, regular expressions can be simplified. Using the pure definition of regular expressions to express the language {banana,nab}* we would be forced to write something nasty like

$$((((b)(a))(n))(((a)(n))(a)) \cup (((n)(a))(b)))*$$

- Using the operator precedence ordering $*, \bullet, \cup$ and the associativity of $\bullet$ allows us to obtain the simpler:

$$(banana \cup nab)*$$

- This is done in the same way as one would simplify the *algebraic* expression with re-ordering disallowed:

$$((((b)(a))(n))(((a)(n))(a))+(((n)(a))(b)))^4 = (banana+nab)^4$$

Regular Expressions: Example

- Question: Find a regular expression that generates the language consisting of all bit-strings which contain a streak of seven 0's or contain two disjoint streaks of three 1's.
  - Legal: 010000000011010, 01110111001, 111111
  - Illegal: 11011010101, 10011111001010, 00000100000

- Answer: $(0 \cup 1)*(0^7 \cup 1^3(0 \cup 1)*1^3)(0 \cup 1)*$
  - An even briefer valid answer is: $\Sigma*(0^7 \cup 1^3 \Sigma *1^3)\Sigma*$
  - The *official* answer using only the standard regular operations is:
    $$(0 \cup 1)*(0000000 \cup 111(0 \cup 1)*111)(0 \cup 1)*$$
  - A brief UNIX answer is:
    ```
    (0|1)*(0{7}|1{3}(0|1)*1{3})(0|1)*
    ```

Regular Expressions: Examples

1) 0*10*

2) $(\Sigma\Sigma)*$

3) 1*∅

4) $\Sigma = \{0,1\}$, {w | w has at least one 1}

5) $\Sigma = \{0,1\}$, {w | w starts and ends with the same symbol}

6) {w | w is a numerical constant with sign and/or fractional part}
   - E.g. 3.1415, -.001, +2000

Regular Expressions: A different view…

- Regular expressions are just strings. Consequently, the set of all regular expressions is a set of strings, so by definition is a language.

- Question: Supposing that only union, concatenation and Kleene-* are considered. What is the alphabet for the language of regular expressions over the base alphabet Σ ?

- Answer: $\Sigma \cup \{ (,), \cup, * \}$

REX → NFA

- Since NFA's are closed under the regular operations we immediately get

- Theorem: Given any regular expression r there is an NFA N which simulates r. That is, the language accepted by N is precisely the language generated by r so that $L(N) = L(r)$. Furthermore, the NFA is constructible in linear time.
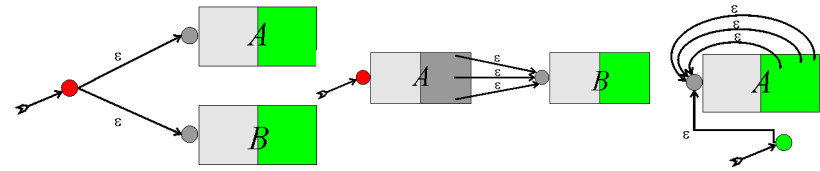
REX → NFA

- *Proof*: The proof works by induction, using the recursive definition of regular expressions. First we need to show how to accept the base case regular expressions $a \in \Sigma$, ε and $\varnothing$. These are respectively accepted by the NFA's:
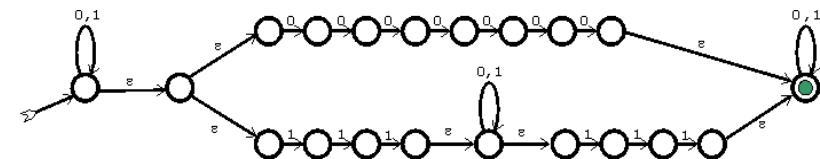

- Finally, we need to show how to inductively accept regular expressions formed by using the regular operations. These are just the constructions that we saw before, encapsulated by:

REX → NFA exercise: Find NFA for $(ab \cup a)^*$

REX → NFA: Example

- Question: Find an NFA for the regular expression

$$(0 \cup 1)^*(0000000 \cup 111(0 \cup 1)^*111)(0 \cup 1)^*$$

of the previous example.

REX → NFA → FA ?!?

- The fact that regular expressions can be converted into NFA's means that it makes sense to call the languages accepted by NFA's "regular."

- However, the regular languages were defined to be the languages accepted by FA's, which are by default, *deterministic*. It would be nice if NFA's could be "determinized" and converted to FA's, for then the definition of "regular" languages, as being FA-accepted would be justified.

- Let's try this next.

NFA's have 3 types of non-determinism

Nondeterminism type	Machine Analog	δ-function	Easy to fix?	Formally
Under-determined	Crash	No output	yes, fail-state	$|\delta(q,a)| = 0$
Over-determined	Random choice	Multi-valued	no	$|\delta(q,a)| > 1$
ε	Pause reading	*Redefine alphabet*	no	$|\delta(q,\varepsilon)| > 0$

Determinizing NFA's: Example

- Idea: We might keep track of all parallel active states as the input is being called out. If at the end of the input, one of the active states happened to be an accept state, the input was accepted.

- Example, consider the following NFA, and its deterministic FA.
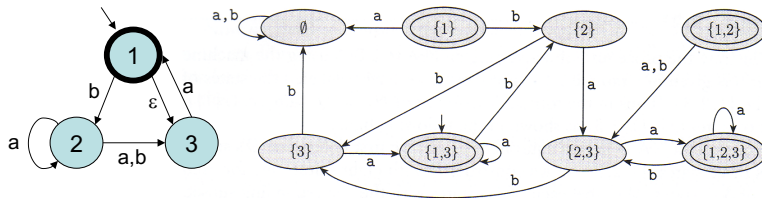
One-Slide-Recipe to Derandomize

- Instead of the states in the NFA, we consider the power-states in the FA. (If the NFA has n states, the FA has 2^n states.)

- First we figure out which power-states will reach which power-states in the FA. (Using the rules of the NFA.)

- Then we must add all epsilon-edges: We redirect pointers that are initially pointing to power-state {a,b,c} to power-state {a,b,c,d,e,f}, if and only if there is an epsilon-edge-only-path pointing from any of the states a,b,c to states d,e,f (a.k.a. transitive closure). We do the very same for the starting state: starting state of FA = {starting state of NFA, all NFA states that can recursively be reached from there}

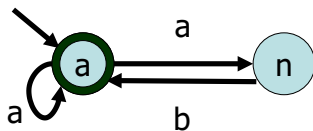- Accepting states of the FA are all states that include a accepting NFA state.

Remarks

- The previous recipe can be made totally formal. More details can be found in the reading material.

- Just following the recipe will often produce a too complicated FA. Sometimes obvious simplifications can be made. In general however, this is not an easy task.

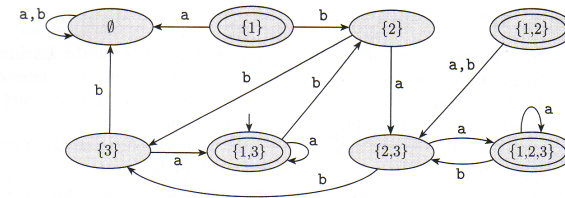## Automata Simplification

- The FA can be simplified. States {1,2} and {1}, for example, cannot be reached. Still the result is not as simple as the NFA.

Derandomization Exercise

- Exercise: Let's derandomize the simplifed two-state NFA from slide 1/70 which we derived from regular expression $(ab \cup a)^*$
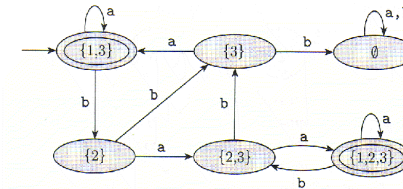
REX → NFA → FA

- Summary: Starting from any NFA, we can use subset construction and the epsilon-transitive-closure to find an equivalent FA accepting the same language. Thus,

- Theorem: If L is any language accepted by an NFA, then there exists a constructible [deterministic] FA which also accepts L.

- Corollary: The class of regular languages is closed under the regular operations.

- Proof: Since NFA's are closed under regular operations, and FA's are by default also NFA's, we can apply the regular operations to any FA's and determinize at the end to obtain an FA accepting the language defined by the regular operations.
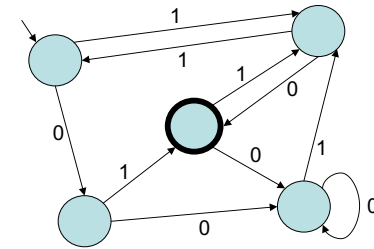
REX → NFA → FA → REX …

- We are one step away from showing that FA's ≈ NFA's ≈ REX's; i.e., all three representation are equivalent. We will be done when we can complete the circle of transformations:
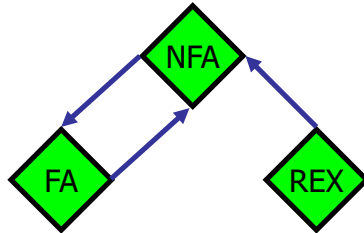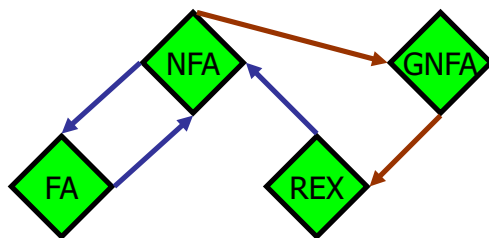
NFA → REX is simple?!?

- Then FA → REX even simpler!

- Please solve this simple example:

REX → NFA → FA → REX …

- In converting NFA's to REX's we'll introduce the most generalized notion of an automaton, the so called "Generalized NFA" or "GNFA". In converting into REX's, we'll first go through a GNFA:

GNFA's

- Definition: A generalized nondeterministic finite automaton (GNFA) is a graph whose edges are labeled by regular expressions,
  - with a unique start state with in-degree 0, but arrows to every other state
  - and a unique accept state with out-degree 0, but arrows from every other state (note that accept state ≠ start state)
  - and an arrow from any state to any other state (including self).

- A GNFA accepts a string s if there exists a path p from the start state to the accept state such that w is an element of the language generated by the regular expression obtained by concatenating all labels of the edges in p.

- The language accepted by a GNFA consists of all the accepted strings of the GNFA.

GNFA Example
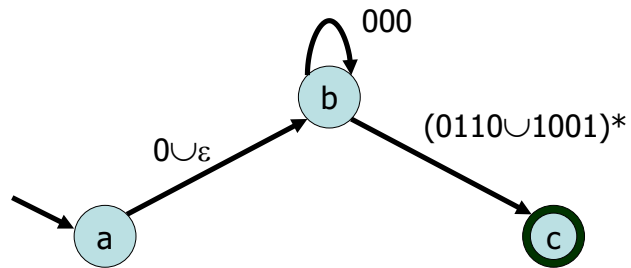
- This is a GNFA because edges are labeled by REX's, start state has no in-edges, and the *unique* accept state has no out-edges.

- Convince yourself that 000000100101100110 is accepted.

NFA → REX conversion process

1. Construct a GNFA from the NFA.
   A. If there are more than one arrows from one state to another, unify them using "∪"
   B. Create a unique start state with in-degree 0
   C. Create a unique accept state of out-degree 0
   D. [If there is no arrow from one state to another, insert one with label Ø]

2. Loop: As long as the GNFA has strictly more than 2 states:
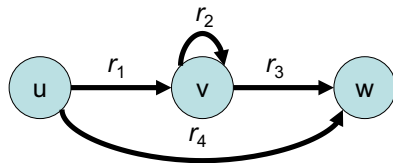   Rip out arbitrary interior state and modify edge labels.
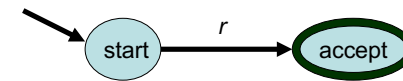

3. The answer is the unique label r.

NFA → REX: Ripping Out.

- Ripping out is done as follows. If you want to rip the middle state v out (for all pairs of neighbors u,w)…
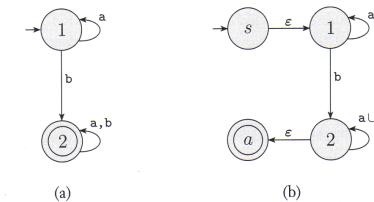

- … then you'll need to recreate all the lost possibilities from u to w. I.e., to the current REX label r_4 of the edge (u,w) you should add the concatenation of the (u,v) label r_1 followed by the (v,v)-loop label r_2 repeated arbitrarily, followed by the (v,w) label r_3. The new (u,w) substitute would therefore be:
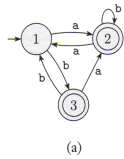
FA → REX: Example

FA → REX: Exercise


(a)
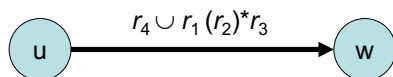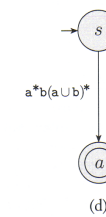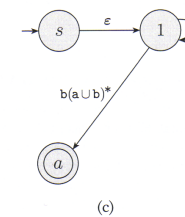
Summary: FA ≈ NFA ≈ REX

- This completes the demonstration that the three methods of describing regular languages are:

1. Deterministic FA's
2. NFA's
3. Regular Expressions

- We have learnt that all these are equivalent.

Remark about Automaton Size

- Creating an automaton of small size is often advantageous.
  - Allows for simpler/cheaper hardware, or better exam grades.
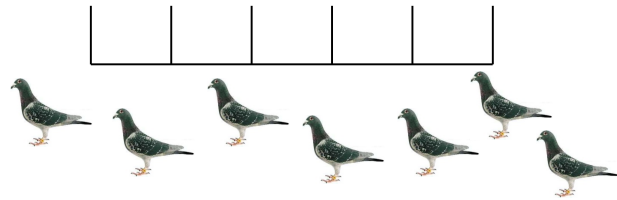 - Designing/Minimizing automata is therefore a funny sport. Example:
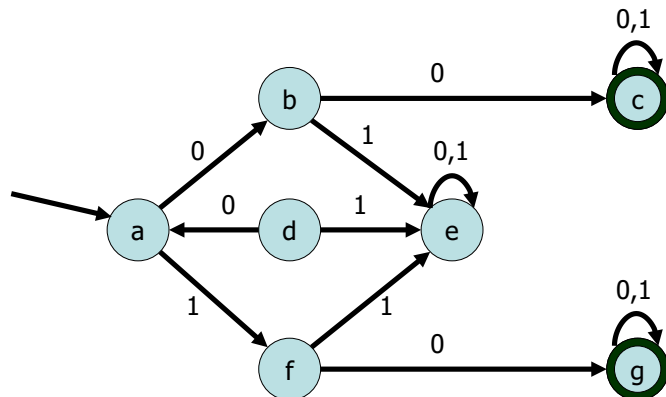
Minimization

- Definition: An automaton is irreducible if
  - it contains no useless states, and
  - no two distinct states are equivalent.

- By just following these two rules, you can arrive at an "irreducible" FA. Generally, such a local minimum does not have to be a global minimum.

- It can be shown however, that these minimization rules actually produce the global minimum automaton.

- The idea is that two prefixes u,v are indistinguishable iff for all suffixes x, ux ∈ L iff vx ∈ L. If u and v are distinguishable, they cannot end up in the same state. Therefore the number of states must be at least as many as the number of pairwise distinguishable prefixes.

Pigeonhole principle

- Consider language L, which contains word $w \in L$.
- Consider an FA which accepts L, with $n < |w|$ states.
- Then, when accepting w, the FA must visit at least one state twice.

- This is according to the pigeonhole (a.k.a. Dirichlet) principle:
  - If m>n pigeons are put into n pigeonholes, there's a hole with more than one pigeon.
  - That's a pretty fancy name for a boring observation…

Languages with unbounded strings

- Consequently, regular languages with unbounded strings can only be recognized by FA (finite! bounded!) automata if these long strings loop.

- The FA can enter the loop once, twice, …, and not at all.
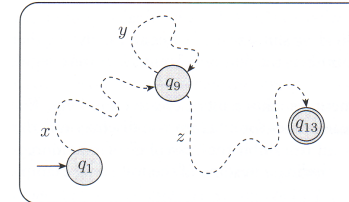- That is, language L contains all $\{xz, xyz, xy^2z, xy^3z, …\}$.

Pumping Lemma

- Theorem: Given a regular language L, there is a number p (called the pumping number) such that any string in L of length $\geq p$ is pumpable within its first p letters.

- In other words, for all $u \in L$ with $|u| \geq p$ we can write:
  - $u = xyz$ (x is a prefix, z is a suffix)
  - $|y| \geq 1$ (mid-portion y is non-empty)
  - $|xy| \leq p$ (pumping occurs in first p letters)
  - $xy^iz \in L$ for all $i \geq 0$ (can pump y-portion)

- If, on the other hand, there is no such p, then the language is not regular.

Pumping Lemma Example

- Let L be the language $\{0^n1^n \mid n \geq 0\}$

- Assume (for the sake of contradiction) that L is regular
- Let p be the pumping length. Let u be the string 0^p1^p.
- Let's check string u against the pumping lemma:

- "In other words, for all $u \in L$ with $|u| \geq p$ we can write:
  - $u = xyz$ (x is a prefix, z is a suffix)
  - $|y| \geq 1$ → y = 0^+ (mid-portion y is non-empty)
  - $|xy| \leq p$ (pumping occurs in first p letters)
  - $xy^iz \in L$ for all $i \geq 0$ (can pump y-portion)"

  → Then, xz or xyyz is not in L. Contradiction!

Let's make the example a bit harder…

- Let L be the language {w | w has an equal number of 0s and 1s}

- Assume (for the sake of contradiction) that L is regular
- Let p be the pumping length. Let u be the string 0^p1^p.
- Let's check string u against the pumping lemma:

- "In other words, for all $u \in L$ with $|u| \geq p$ we can write:
  - $u = xyz$ (x is a prefix, z is a suffix)
  - $|y| \geq 1$ (mid-portion y is non-empty)
  - $|xy| \leq p$ (pumping occurs in first p letters)
  - $xy^iz \in L$ for all $i \geq 0$ (can pump y-portion)"

Harder example continued

- Again, y must consist of 0s only!
- Pump it there! Clearly again, if xyz $\in$ L, then xz or xyyz are not in L.

- There's another alternative proof for this example:
  - 0*1* is regular.
  - $\cap$ is a regular operation.
  - If L regular, then L $\cap$ 0*1* is also regular.
  - However, L $\cap$ 0*1* is the language we studied in the previous example (0^n1^n). A contradiction.

Now you try…

- Is $L_1 = \{ww \mid w \in (0 \cup 1)^*\}$ regular?

- Is $L_2 = \{1^n \mid n \text{ being a prime number }\}$ regular?