91.304 Foundations of (Theoretical) Computer Science

Chapter 1 Lecture Notes (Section 1.2: NFA's)

David Martin
dm@cs.uml.edu

With some modifications by Prof. Karen Daniels
Slides also added from http://cis.k.hosei.ac.jp/~yukita/ in some places.

SOME RIGHTS RESERVED

This work is licensed under the Creative Commons Attribution-ShareAlike License. To view a copy of this license, visit http://creativecommons.org/licenses/by-sa/2.0/ or send a letter to Creative Commons, 559 Nathan Abbott Way, Stanford, California 94305, USA.

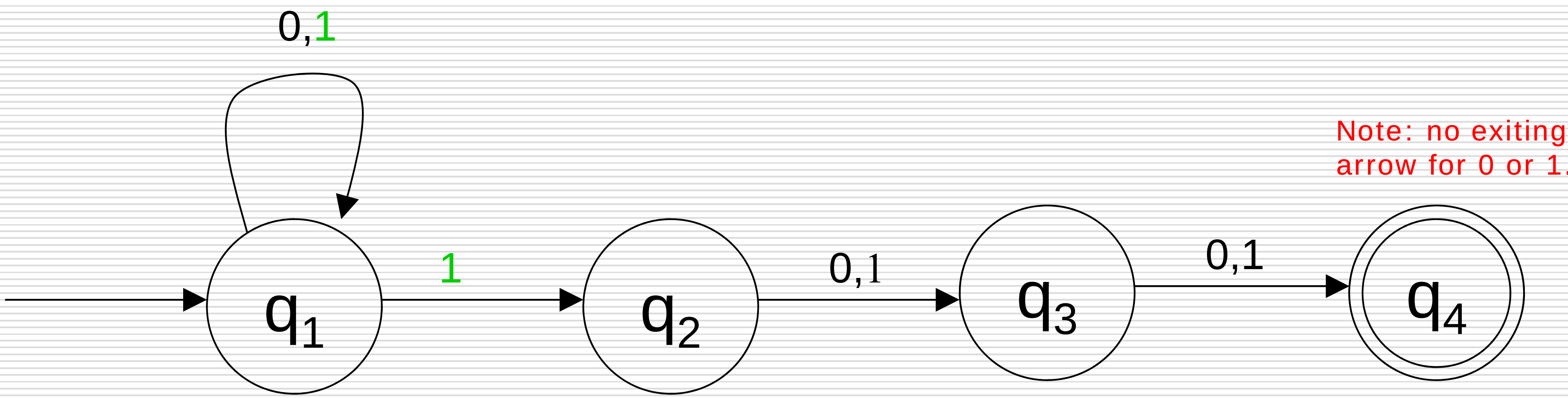
Nondetermistic Finite Automata

- A nondeterministic finite automaton can be different from a deterministic one in that
  - for any input symbol, nondeterministic one can transit to more than one state.
  - epsilon transition (ε), which “consumes” no input symbols
- NFA and DFA stand for *nondeterministic finite automaton* and *deterministic finite automaton*, respectively.
- NFAs and DFAs are equally powerful, but NFA adds *notational* power that can simplify descriptions.
  - Example: L&P

Nondeterministic Finite Automata

- Will relax two of these DFA rules:
  1. Each (state, char) input must produce exactly one (state) output
  2. Must consume one character in order to advance state
- The NFA accepts the input if **there exists** any way of reading the input that winds up in an accepting state at the end of the string
  - Otherwise it rejects the input

Example: NFA N_2

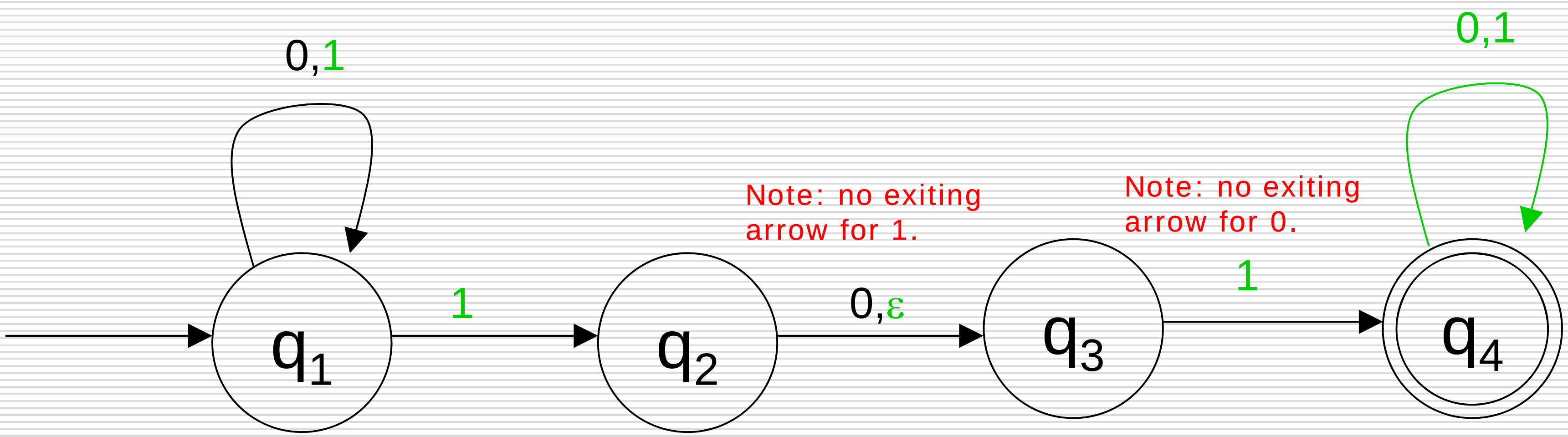


Let language *A* consist of all strings over {0,1} containing a 1 in the third position from the end. N_2 recognizes *A*.

Note: Multiple choice on input 1 from state q_1 makes this an NFA.

Later we show a DFA equivalent to this NFA using construction of Thm. 1.39.

NFA N_1



Now introduce ε.
What language does this NFA accept?

variation on http://cis.k.hosei.ac.jp/~yukita/

NFA N_1 Execution on input 010110

Source: Sipser Textbook

Let's consider some sample runs of the NFA N_1 shown in Figure 1.27. The computation of N_1 on input 010110 is depicted in the following figure.

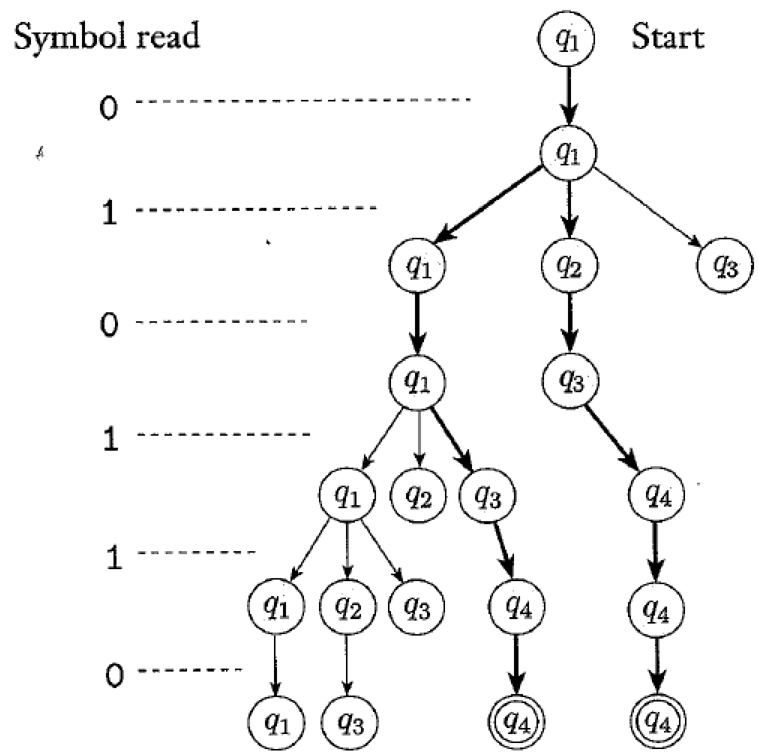


FIGURE 1.29
The computation of N_1 on input 010110

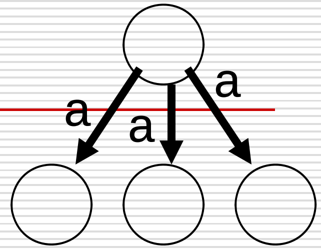
Note pictorial "jump" on ε to next state. This varies slightly from transition function depiction on p. 54.

How does N_1 behave on input 01001?

Ways to think of NFAs

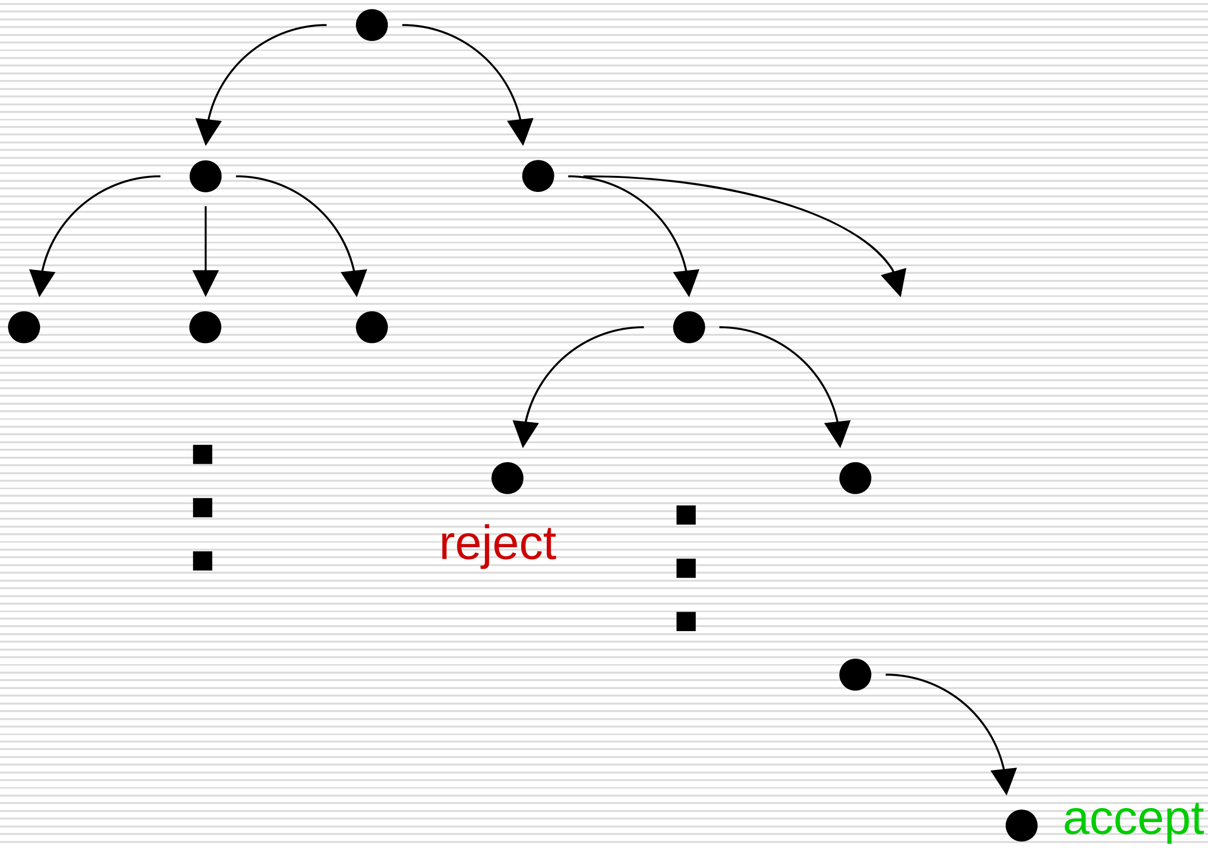
- NFAs **want** to accept inputs and will always take the most advantageous alternative(s)
  - Because they will accept if there exists **any** way to get to an accepting state at the end of the string
  - The quickest way there may be just one of many ways, but it doesn’t matter

Ways to think of NFAs



- fork() model
  - Input string is in a variable
  - fork() at every nondeterministic choice point
    - subprocess 1 (parent) follows first transition
    - subprocess 2 (child) follows second
    - subprocess 3 (child) follows third (if any), etc.
  - A process that can't follow any transition calls exit() -- and gives up its ability to accept
  - A process that makes it through the whole string and is in an accepting state prints out "ACCEPT"
    - A single ACCEPT is enough

Parallel world and NFA



variation on http://cis.k.hosei.ac.jp/~yukita/

Syntax of DFA (repeat)

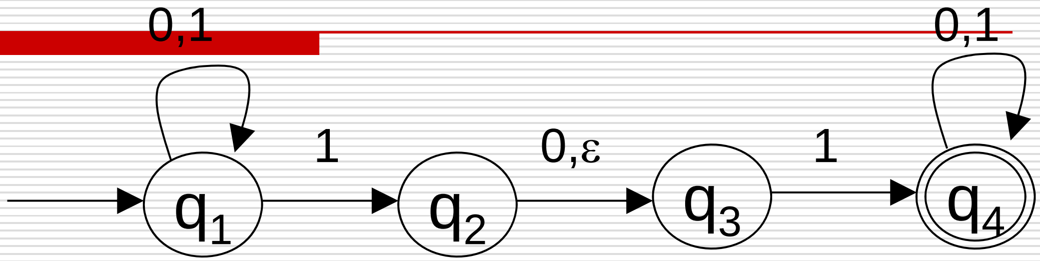
- A **deterministic finite automaton (DFA)** is a 5-tuple (Q,Σ,δ,q_0,F) such that
  1. Q is a *finite* set of states
  2. Σ is an alphabet
  3. $\delta: Q \times \Sigma \to Q$ is the transition function
  4. $q_0 \in Q$ is the start state
  5. $F \subseteq Q$ is the set of accepting states
- Usually these names are used, but others are possible as long as the role is clear

Syntax of NFA

- A **nondeterministic finite automaton (NFA)** is a 5-tuple (Q,Σ, δ,q_0,F) such that
  1. Q is a *finite* set of states
  2. Σ is an alphabet
  3. $\delta: Q\times(\Sigma\cup\{\varepsilon\})\rightarrow\mathcal{P}(Q)$ is the transition function
  4. $q_0 \in Q$ is the start state
  5. $F \subseteq Q$ is the set of accepting states
- Usually these names are used, but others are possible as long as the role is clear

Note: $\Sigma\varepsilon = \Sigma \cup \{\varepsilon\}$ (see p. 53)

NFA N_1 (again) p. 54



1. $Q = \{q_1, q_2, q_3, q_4\}$,
2. $\Sigma = \{0,1\}$
3. δ is given as

	0	1	ε
q_1	$\{q_1\}$	$\{q_1, q_2\}$	$\varnothing$
q_2	$\{q_3\}$	$\varnothing$	$\{q_3\}$
q_3	$\varnothing$	$\{q_4\}$	$\varnothing$
q_4	$\{q_4\}$	$\{q_4\}$	$\varnothing$

Note the use of **sets** here in contrast to DFA.

4. q_1 is the start state.
5. $F = \{q_4\}$.

Board work: Resolve transition table with Figure 1.29 for ε.

The Subset Construction

Theorem 1.39 For every NFA M_1 there exists a DFA M_2 such that $L(M_1) = L(M_2)$.

Corollary 1.40 A language is REGular if and only if some nondeterministic finite automaton recognizes it.

The Subset Construction

Proof: Let N=(Q,Σ,δ,q_0,F) be the NFA and define the DFA M=(Q',Σ,δ', q_0',F') as follows:

1. Q' = $\mathcal{P}$(Q).
   - Each state of the DFA records the set of states that the NFA can simultaneously be in
   - Can compare DFA states for equality but also look "inside" the state name to find a set of NFA state names
2. Define:

   E(R) = {q|q is reachable from R via ε arrow(s)}

$$\delta'(\mathrm{R},a) = \{q \in \mathrm{Q} \mid q \in \mathrm{E}(\delta(r,a)) \text{ for some } r \in \mathrm{R}\}$$

Go to whatever states are reachable from the states in R and reading the character ***a***

Remember: in an NFA,
δ: Q $\times\Sigma_\varepsilon \rightarrow \mathcal{P}$(Q) from def

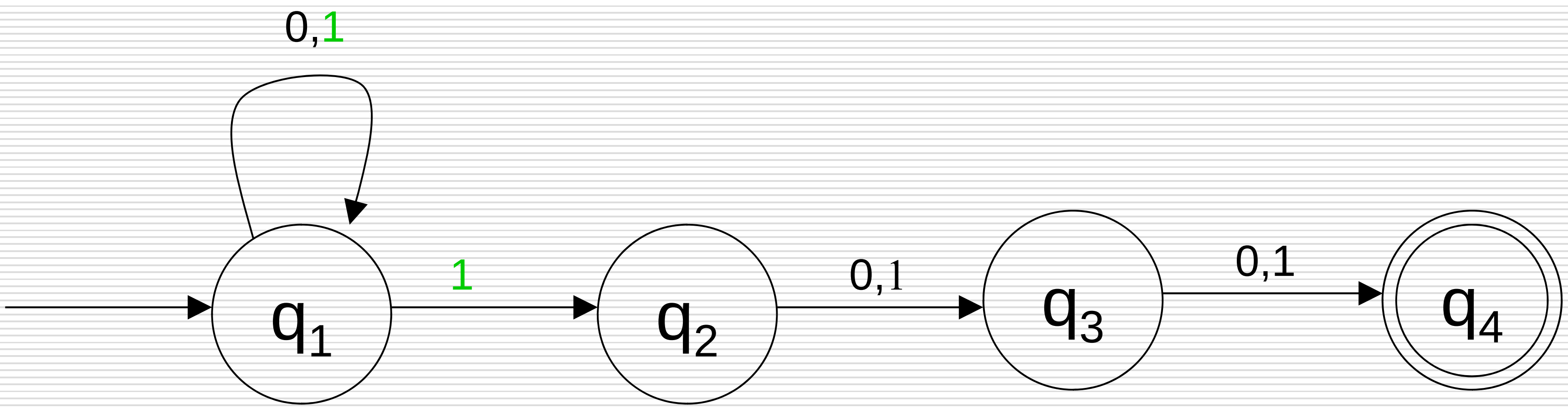
The Subset Construction

3. $q_0' = E(\{q_0\})$
4. $F' = \{R \in Q' \mid R$ contains an accept state of $N\}$

The effect is that the DFA knows all states that are reachable in the NFA after reading the string so far. If any one of them is accepting, then the current DFA state is accepting too, otherwise it's not.

If you believe this then that's all it takes to see that the construction is correct. So, convince yourself with an example. **QED**

Example: NFA N_2 (again)

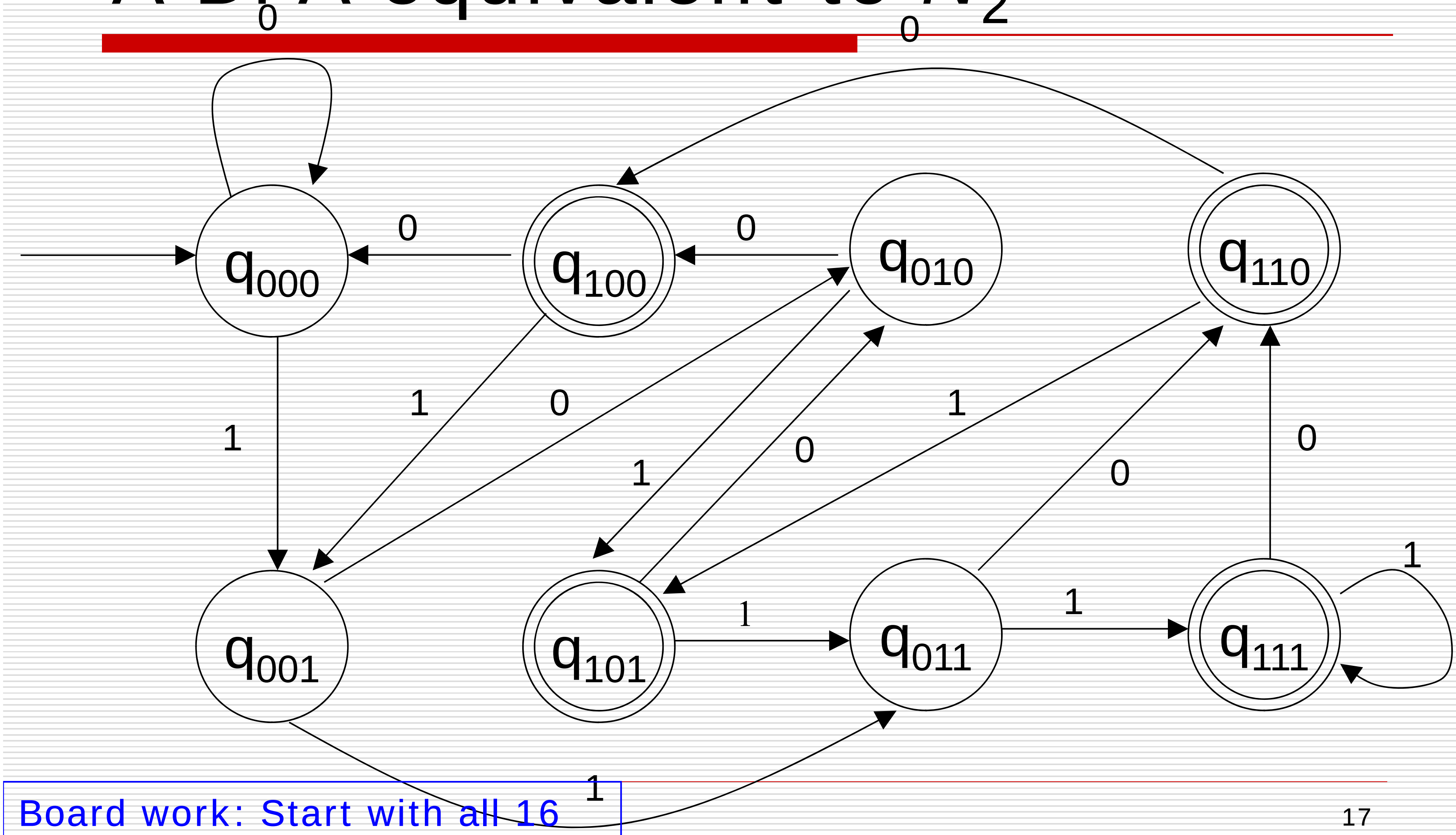


Let language A consist of all strings over {0,1} containing a 1 in the third position from the end. N_2 recognizes A.

No ε's in this example.

variation on http://cis.k.hosei.ac.jp/~yukita/

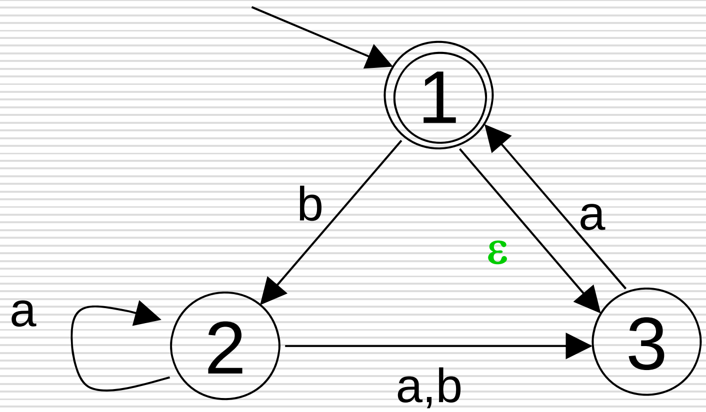
A DFA equivalent to N_2



Board work: Start with all 16 states, including unreachables.

Variation on http://cis.k.hosei.ac.jp/~yukita/

Example 1.41 NFA N_4 to DFA

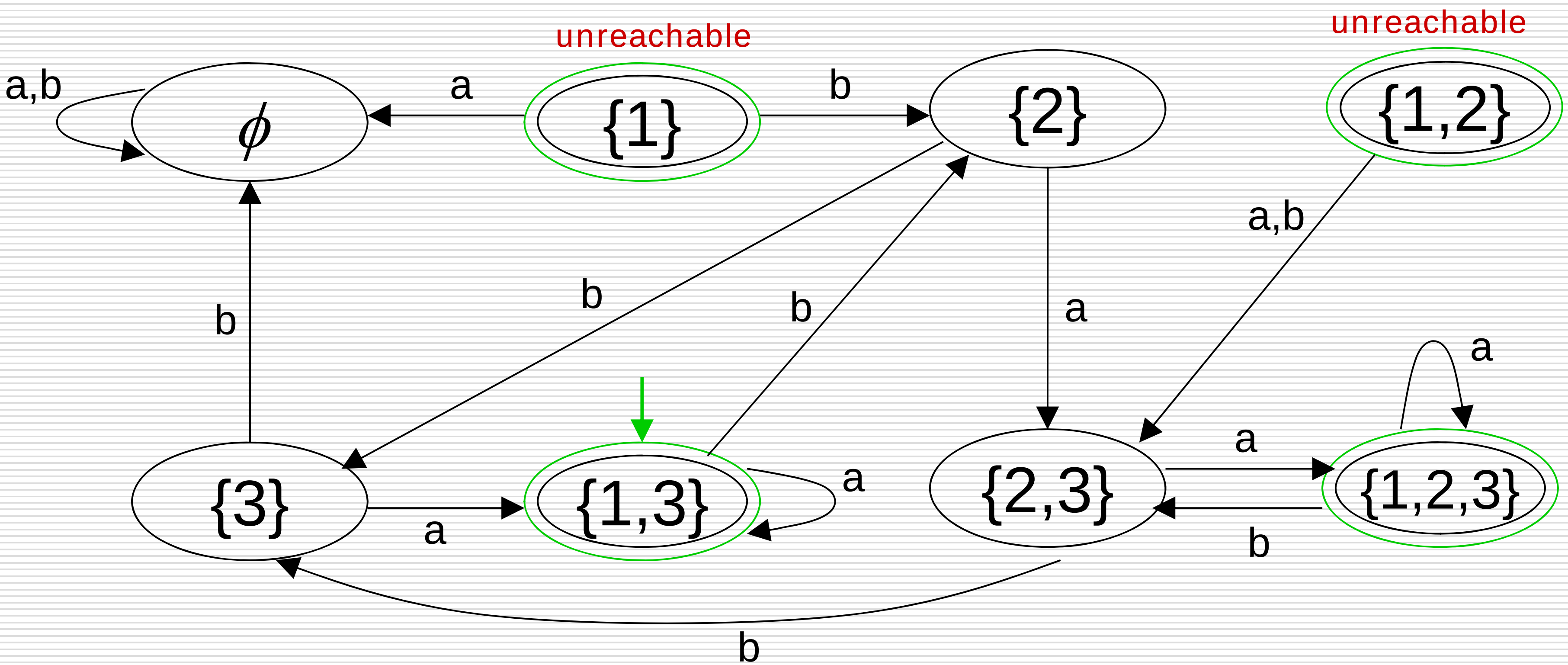


Given $N_4 = \{\{1,2,3\},\{a,b\},\delta,1,\{3\}\}$, we want to construct an equivalent DFA D. D's state set may be expressed as

$2^{\{1,2,3\}} = \{\emptyset,\{1\},\{2\},\{3\},\{1,2\},\{1,3\},\{2,3\},\{1,2,3\}\}$.

variation on http://cis.k.hosei.ac.jp/~yukita/

The state diagram of *D*



Done in textbook.

variation on http://cis.k.hosei.ac.jp/~yukita/

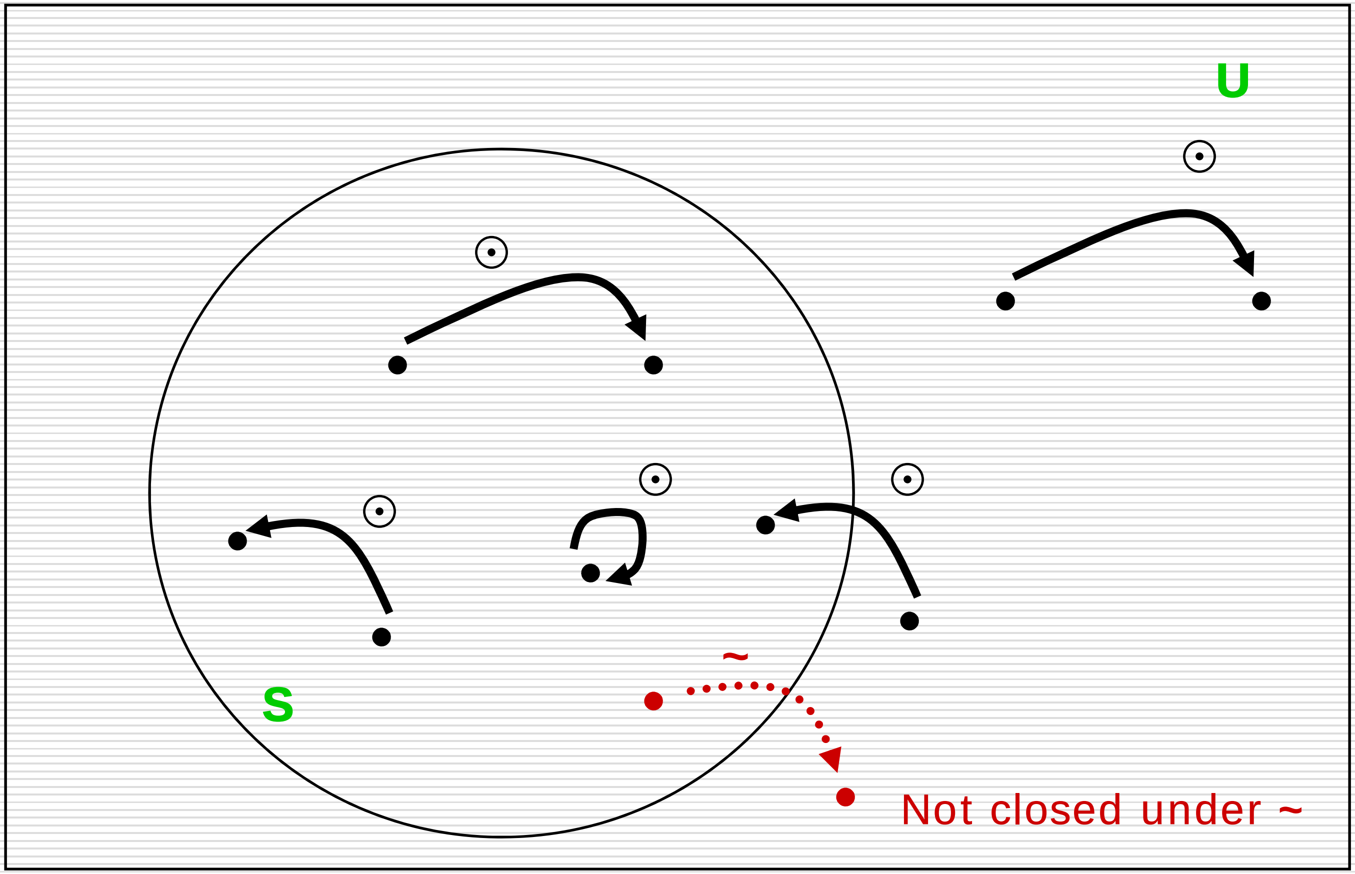
Subset construction conclusion

- Adding nondeterminism makes programs shorter but not able to do new things
- Remember: regular languages are **defined** to be those "recognized by a DFA"
- We now have a **result** that says that every language that is recognized by an NFA is regular too
  - So if you are asked to show that a language is regular, you can exhibit a DFA **or** NFA for it and rely on the subset construction theorem
  - Sometimes questions are specifically about DFAs or NFAs, though... pay attention to the precise wording

Closure properties

- The presence or absence of closure properties says something about how well a set tolerates an operation
- **Definition**. Let $S \subseteq U$ be a set in some universe U and $\odot$ be an operation on elements of U. We say that **S is closed under** $\odot$ if applying $\odot$ to element(s) of S produces another element of S.
  - For example, if $\odot$ is a binary operation $\odot: U \times U \rightarrow U$, then we're saying that $(\forall\, x \in S \text{ and } y \in S)\; x \odot y \in S$

Closure properties illustrated



Applying the ⊙ operation to elements of S never takes you ouside of S.

S is **closed** with respect to ⊙

This example shows *unary* operations

More examples

- $L_1 = \{x \in \{0,1\}^* : |x| \text{ is a multiple of } 3\}$
  - is closed under string reversal and concatenation
- $L_3 = \{x \in \{0,1\}^* \mid \text{the binary number } x \text{ is a multiple of } 3\}$
  - is also closed under string reversal and concatenation, harder to see though
- $L_4 = \{x \in \{a,b\}^* \mid x \text{ contains an odd \# of 'b's and an even \# of 'a's}\}$
  - is closed under string reversal
  - is not closed under string concatenation

Closure: higher abstraction

- We will usually be concerned with closure of **language classes** under language operations
  - Previous examples were closure of sets containing *non-set* elements under various familiar operations
  - We consider DFAs and NFAs to be *programs* and we want assurance that their outputs can be combined in desired ways just by manipulating their programs (like using one as a subroutine for the other)
  - Representative question: is REG closed under (language) concatenation?

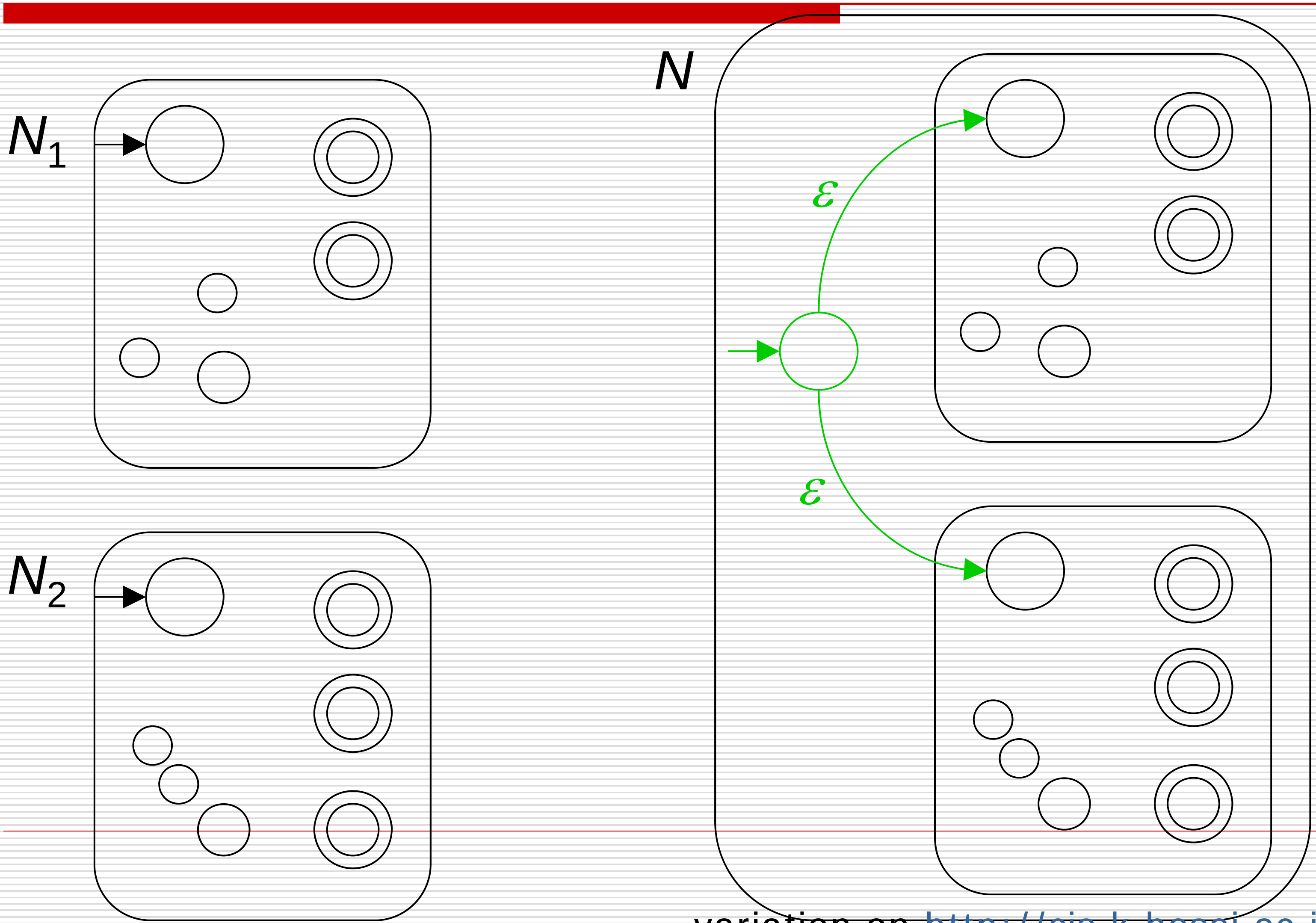
The regular operations

- The **regular operations** on languages are
  - $\cup$ (union)
  - $\cdot$ (concatenation)
  - $*$ (Kleene star)
- The name "regular operations" is not that important
  - Too bad we use the word "regular" for so much
- REG is closed under these regular operations
  - That's why they're called "regular" operations
  - This does **not** mean that each regular language is closed under each of these operations!

The regular operations

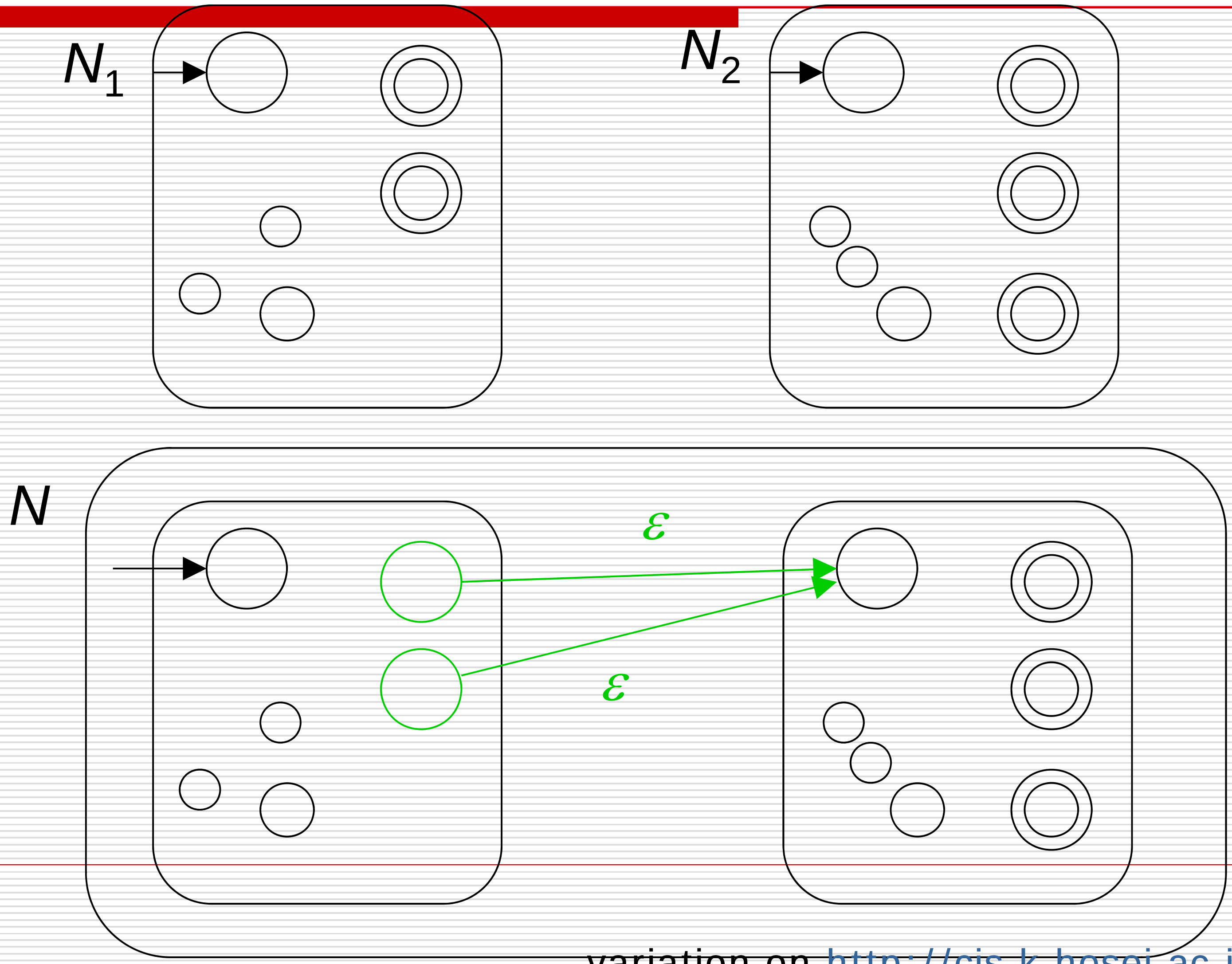
- REG is closed under union: Theorem 1.25 (using DFAs), Theorem 1.45 (using NFAs)
- REG is closed under concatenation: Theorem 1.47 (NFAs)
- REG is closed under $*$: Theorem 1.49 (NFAs)
- **Study these constructions!!**
- REG is also closed under complement, intersection and reversal (not in book)

Theorem 1.45 The class of regular languages is closed under the union operation.



variation on http://cis.k.hosei.ac.jp/~yukita/

Theorem 1.47 The class of regular languages is closed under the concatenation operation.



variation on http://cis.k.hosei.ac.jp/~yukita/

Theorem 1.24 The class of regular languages is closed under the star operation.