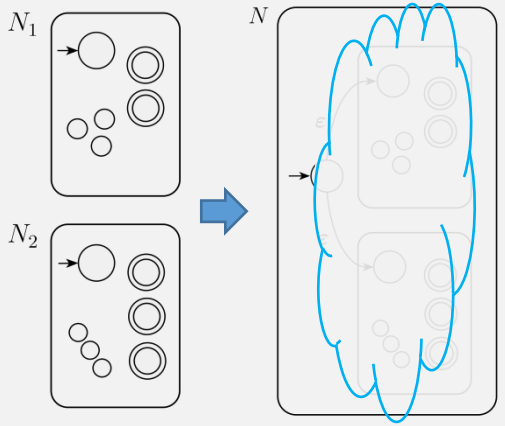
CS 420

Combining DFAs and Closed Operations

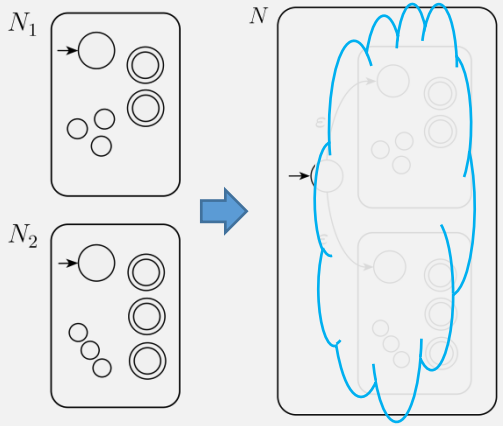
Thursday, September 24, 2025

UMass Boston Computer Science



Announcements

- HW 1 in
  - ~~Due: Tue 9/22~~
- HW 2 out
  - Due: Tue 9/29 12:30pm
- Check previous Piazza posts before posting!
- All questions about languages or machines must use examples!



HW 1 Observations

- Problems must be assigned to the correct pages in GradeScope
- Proofs must follow this semester's requirements, e.g., **constructors** and **statements** and **Justifications** table
- Questions / Complaints?
  - Open GradeScope re-grade request
  - Do not ask the instructor or TA (they probably didn't grade it)!

Common mistakes

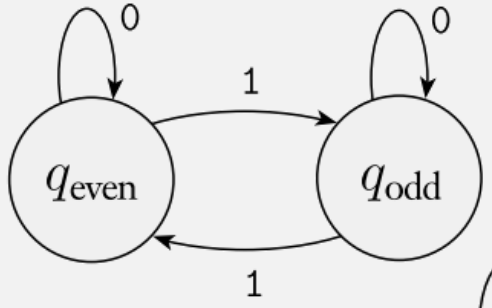
- Extraneous defs,
  - $q_0=q_0$
  - $Q = \{1,2,3\}$
    - but no use of Q
  - $M = (Q,\Sigma,\delta,q_0,F)$
    - with unbound vars!
    - And no `mk-DFA` constructor
- Wrong types!
  - $M =$ `mk-DFA` $\{Q,\Sigma,\delta,q_0,F\}$
    - Or some other non-tuple
    - Or forgot / omitted the tuple
  - $F = q_3$
  - $\Sigma=\{\text{“1”}, \text{”2”}\}$

Last Time

Design a DFA: accept strs with odd # 1s

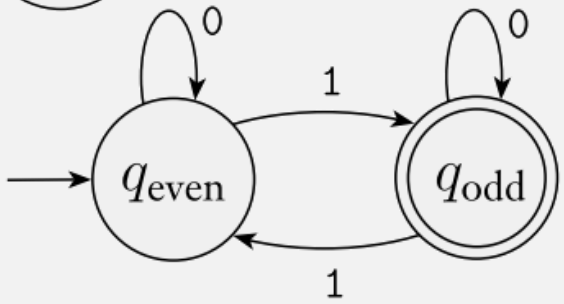
- States:
  - 2 states:
    - seen even 1s so far
    - seen odds 1s so far

- Alphabet: 0 and 1
- Transitions:



let StateDiag_1 =

- Start / Accept states:



(from now on: must use "let" or "where" to define variable names)

Proving That a Language is Regular

Prove: L IS-A **Regular** Language, where L = some **Language** { ... }

Proof:

Statements	**Justifications**
1. M IS-A **DFA** Machine	1.
(TODO 1: design M)	
2. M **recognizes** L	2.
3. L IS-A **Regular** Language	3. APPLY IF M IS-A **DFA** Machine AND M **recognizes** L THEN L IS-A **Regular** Language
	TO **mk-AND-proof** $Proof_1$ $Proof_2$

DefRegLang

Proving That a Language is Regular

Prove: L IS-A **Regular Language**, where $L = \{w \mid w$ has **odd** # of **1**s$\}$

Proof:

	Statements	**Justifications**
☑ 1.	StateDiag$_1$ IS-A **DFA Machine**	1. **BY** ~~DefDFA~~ DefDFAStateDiagram
2.	StateDiag$_1$ **recognizes** L	2. (TODO 2: ???)
3.	L IS-A **Regular** Language	3. **APPLY** IF M IS-A **DFA** Machine AND M **recognizes** L THEN L IS-A **Regular** Language

Writing this means **DFA** was **defined** using **mk-DFA**

Alternatively, writing this means **DFA** was **defined** as (an informal) **state diagram**

(only valid if problem allows, and state diagram has all required parts!)

DefDFA

DEFINITION

A *finite automaton* is a mk-DFA $(Q, \Sigma, \delta, q_0, F)$, where

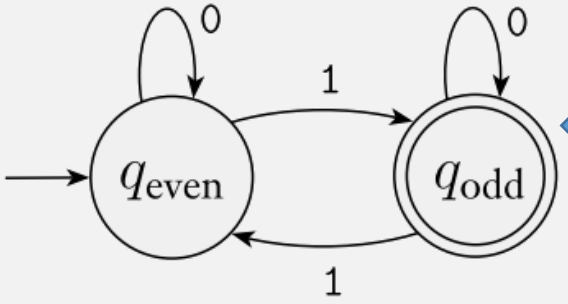
1. Q is a finite set called the *states*,
2. Σ is a finite set called the *alphabet*,
3. $\delta: Q \times \Sigma \longrightarrow Q$ is the *transition function*,
4. $q_0 \in Q$ is the *start state*, and
5. $F \subseteq Q$ is the *set of accept states*.

(must convert state diagram to formal description)

DefRegLang

TO **mk-AND-proof** $Proof_1$ $Proof_2$

“Prove” that DFA recognizes a language



???

Language = { w | w has **odd** # of **1**s}

String	In the language?
1	Yes
0	No
01	Yes
11	No
1101	Yes
ε	no

$\Sigma = \{0,1\}$

Q1: How can we **“prove”** that a **computation** does what it’s **“supposed to do”?**

These kinds of proofs (e.g., HMU 2.3.4) are usually **hard,** sometimes impossible!

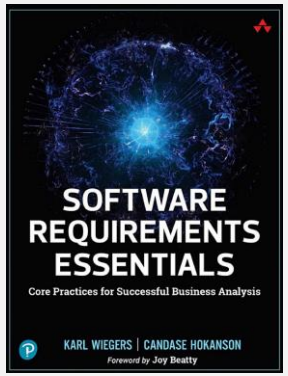
… so we will not do them in this course

But we still need to do something …

Q2: How do **programmers “prove”** that a **computation** does what it’s **“supposed to do”?**

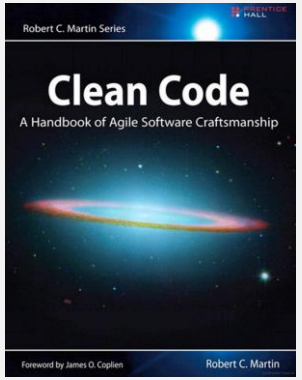
i.e., that the program is **“correct”**?

Interlude: Software Dev 101



Specification (i.e., **requirements**):

- What the **computation**, i.e., **program**, **should do**

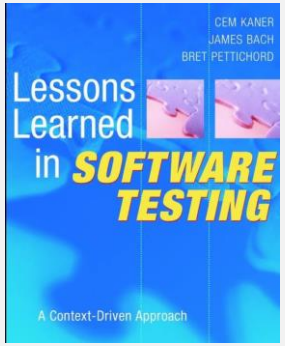


Implementation (i.e., the **code**):

- What the **program does**

Verification (i.e., **testing**):

- Does the **program do what it's supposed to?**
- (i.e., is the **program "correct"**?)



let ExampleTable_1 =

"Prove" that DFA recognizes a language

Verification
(does the **program do what it's supposed to do?**)

These **columns must match** for the **DFA** to **be "correct"!**

Confirm **the DFA:**
- **Accepts strings** in the **language**
- **Rejects strings** <u>not</u> in the **language**

- **In-class exercise** (part 2):

(2 min)

String	In the language?
1	Yes
0	No
01	Yes
11	No
1101	Yes
ε	no

Not a <u>real</u> proof, but …

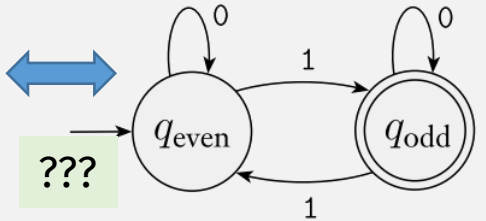
In this class, **a table like this is sufficient to "prove" that a DFA recognizes a language**

Analogous to **what programmers do** (write tests) to **"prove" their computation** (code) **"works"**

Specification
(what the **program should do**)

Implementation
(what the **program does**)

Language = { *w* | *w* has **odd** # of **1**s}



???

Proving That a Language is Regular

Prove: L IS-A **Regular** Language, where $L = \{w \mid w$ has **odd** # of **1**s$\}$

Proof:

Statements	**Justifications**
1. StateDiag_1 IS-A **DFA** Machine	1. **BY** DefDFAStateDiagram
2. StateDiag_1 **recognizes** L	2. (TODO 2: ???)
3. L IS-A **Regular** Language	3. **APPLY** IF M IS-A **DFA** Machine AND M **recognizes** L THEN L IS-A **Regular** Language **TO** **mk-AND-proof** $Proof_1$ $Proof_2$

DefRegLang

Proving That a Language is Regular

Prove: L IS-A **Regular** Language, where $L = \{w \mid w$ has **odd** # of **1**s$\}$

Proof:

	Statements	**Justifications**
☑ 1.	StateDiag_1 IS-A **DFA** Machine	1. **BY** DefDFAStateDiagram
☑ 2.	StateDiag_1 **recognizes** L	2. **BY** ExamplesTable_1 **COMPARING** StateDiag_1, L
☑ 3.	L IS-A **Regular** Language	3. **APPLY** IF M IS-A **DFA** Machine AND M **recognizes** L THEN L IS-A **Regular** Language **TO** **mk-AND-proof** $Proof_1$ $Proof_2$

Not a real proof, but …

In this class, an "Examples Table" is sufficient to "prove" that a Machine recognizes a language

Remember:
You can refer to facts in our "library" by name

DefRegLang

Proving That a Language is Regular

Prove: L IS-A **Regular** Language, where $L = \{w \mid w$ has **odd** # of **1**s$\}$

Proof:

Statements	**Justifications**
1. StateDiag_1 IS-A **DFA** Machine	1. **BY** DefDFAStateDiagram
2. StateDiag_1 **recognizes** L	2. **BY** ExamplesTable_1 **COMPARING** $\text{StateDiag}_1, L$
3. L IS-A **Regular** Language	3. **APPLY** DefRegLang **TO** **mk-AND-proof** $Proof_1$ $Proof_2$

NOTE:
You should write the proof this way

Another In-class exercise

let $A = \{ w \mid w \text{ has exactly three 1's} \}$

- Prove: A IS-A **Regular** Language

- Where **Σ= {0, 1},**

Remember:
To understand the language, always come up with string examples first (in a table)! Both:
- in the language
- and not in the language

You will need this later in the proof anyways!

DEFINITION

A ***finite automaton*** is a `mk-DFA` $(Q, \Sigma, \delta, q_0, F)$, where

1. Q is a finite set called the ***states***,
2. Σ is a finite set called the ***alphabet***,
3. $\delta : Q \times \Sigma \longrightarrow Q$ is the ***transition function***,
4. $q_0 \in Q$ is the ***start state***, and
5. $F \subseteq Q$ is the ***set of accept states***.

Proving That a Language is Regular

Prove: A IS-A **Regular** Language, where $A = \{ w \mid w \text{ has exactly three 1's} \}$

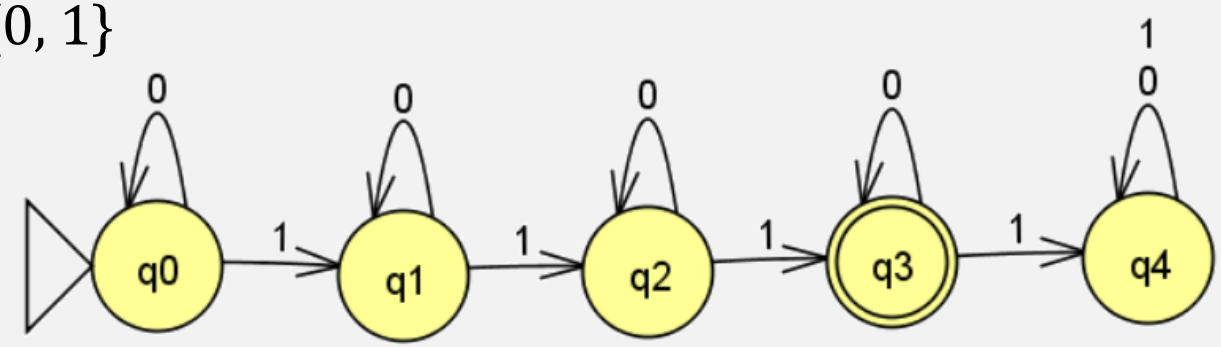
Proof:

Statements	**Justifications**
1. M IS-A **DFA** Machine	1.
(TODO 1: design M)	
2. M **recognizes** A	2.
3. A IS-A **Regular** Language	3. APPLY IF M IS-A **DFA** Machine AND M **recognizes** L THEN L IS-A **Regular** Language (DefRegLang) TO `mk-AND-proof` $Proof_1$ $Proof_2$

In-class exercise Solution

- Design finite automata recognizing:
  $\{w \mid w$ has **exactly three 1**s$\}$

- *States*:
  - Need one state to represent how many 1's seen so far
  - Q = {q0, q1, q2, q3, q4+}
- *Alphabet*: $\Sigma = \{0, 1\}$
- *Transitions*:



- *Start state*:
  - q0
- *Accept states*:
  - {q3}

So: a **DFA's computation** recognizes simple string patterns?

Yes!

Have you ever used **a programming language feature** for recognizing simple string patterns?

Regular Expressions!
(stay tuned!)

Programming Advice: Break Down Complex Problems

2. Break down the problem

Complexity is really just a bunch of simple problems chained together. Try to break down your task into smaller chunks that are more manageable.

`https://dev.to/nuxt-wimadev/7-powerful-principles-to-tackle-complex-coding-problems-40a5`

- Breaking big scary unknown problems into small manageable ones is a **core skill** for developers. And unlike syntax, it can't be easily learned from google. or AI!

This last point should be obvious to anyone who's been coding for a while.

`https://medium.com/@dannysmith/breaking-down-problems-its-hard-when-you-re-learning-to-code-f10269f4ccd5`

2. Break it down

After understanding the problem, the next process is to break down the problem into smaller sub-problems.

`https://javascript.plainenglish.io/5-step-process-to-solve-complex-programming-problems-8e4f74cfd88e`

… and then **combine the** (small) **solutions**

Combining DFA Computation?

("core skill" for programmers)

Password Requirements

- Passwords must have a minimum length of ten (10) characters - but more is better!
- Passwords **must include at least 3** different types of characters:
  - upper-case letters (A-Z)
  - lower-case letters (a-z)
  - symbols or special characters (%, &, *, $, etc.)
  - numbers (0-9)
- Passwords cannot contain all or part of your email address
- Passwords cannot be re-used

DFA

DFA

DFA

DFA

DFA

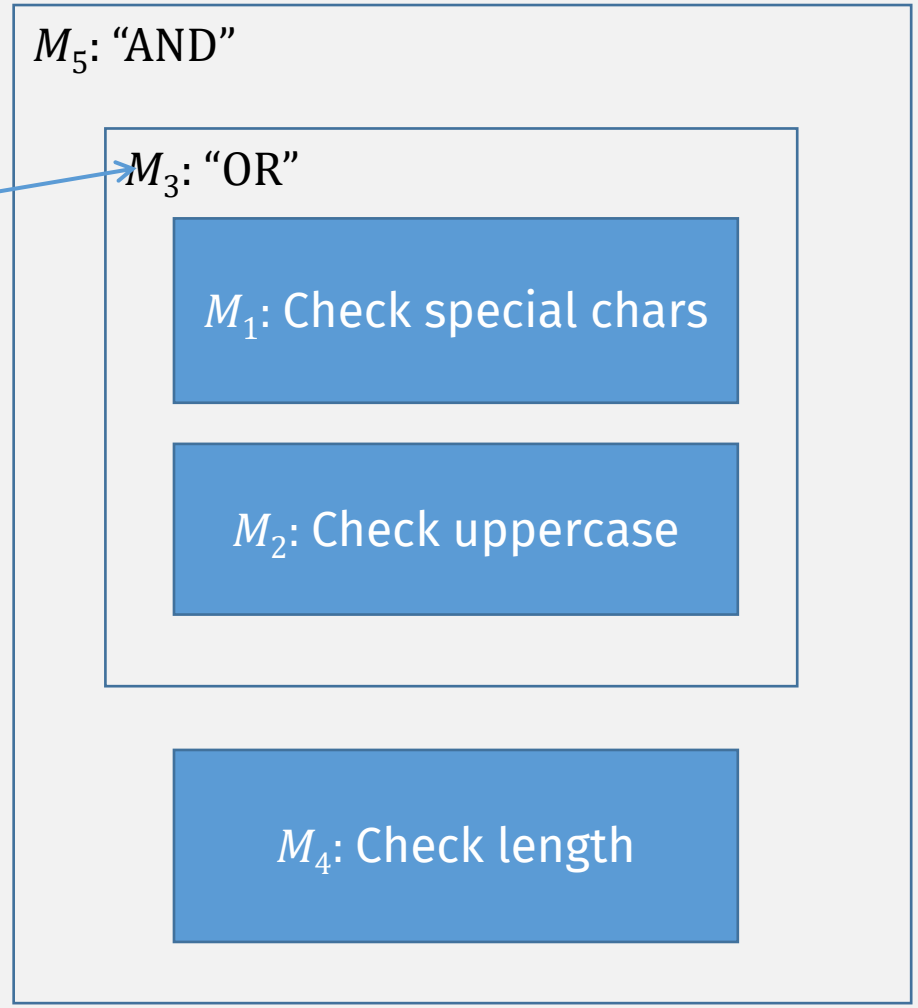
DFA

To match <u>all</u> requirements,
<u>combine</u> smaller DFAs into **one big DFA?**

umb.edu/it/software-systems/password/

Password Checker DFAs

To <u>combine more than once</u>, this must be a **DFA**



Want: to **easily <u>combine</u> DFAs,** i.e., **<u>composability</u>**

<u>We want these operations:</u>

"OR" : **DFA** × **DFA** → **DFA**

"AND" : **DFA** × **DFA** → **DFA**

To <u>combine more than once</u>, **operations** must **preserve input type**, i.e., be **closed!**

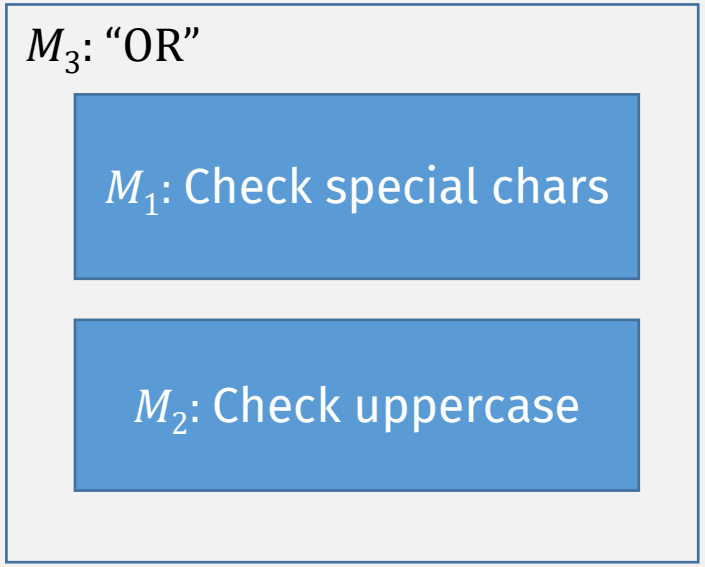
“Closed” Operations

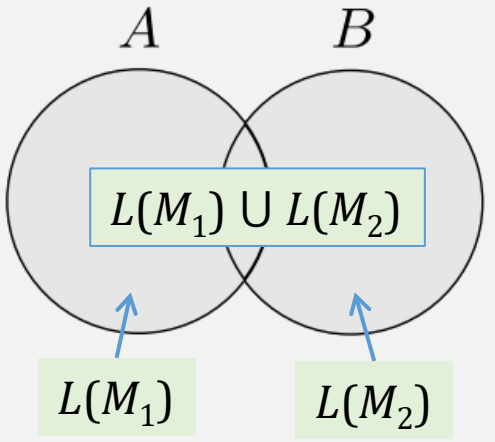
- Set of Natural numbers = {0, 1, 2, …}
  - Closed under addition:
    - if x and y are **Natural numbers,**
    - then $z = x + y$ is a **Natural number**
  - Closed under multiplication?
    - **yes**
  - Closed under subtraction?
    - **no**
- Integers = {…, -2, -1, 0, 1, 2, …}
  - Closed under addition and multiplication
  - Closed under subtraction?
    - **yes**
  - Closed under division?
    - **no**
- Rational numbers = $\{x \mid x = y/z, y$ and z are **Integers**$\}$
  - Closed under division?
    - **No**?
    - **Yes** if z !=0

A **set** is **closed** under an operation if:
the result of applying the operation to **members** of the **set** is in the same set

i.e., **input set**(s) = **output set**

Password Checker: "OR" = "Union"





A **DFA** is **equivalent** to a **language**
(the set of all strings accepted by the DFA)

Combining DFAs is **equivalent** to **combining languages**
(the set of all strings accepted by the DFAs)

U : Language × Language → Language

Union: $A \cup B = \{x \mid x \in A \text{ or } x \in B\}$

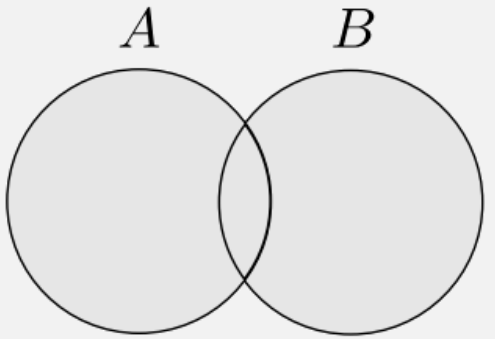
(infix function notation)

Union of Languages

Let the alphabet Σ be the standard 26 letters $\{\texttt{a}, \texttt{b}, \ldots, \texttt{z}\}$.

If $A =$ {**fort**, **south**} $B =$ {**point**, **boston**}

$A \cup B =$ {**fort**, **south**, **point**, **boston**}



Want: "Closed" Ops For Regular Langs!

- Set of Regular Languages = $\{L_1, L_2, \ldots\}$
  - Closed under …?
    - Union ("OR")
    - Intersection ("AND")
    - …

A set is **closed** under an operation if:
the result of applying the operation to
members of the set is in the same set

i.e., input set(s) = output set

For Example:

U : **Regular** Lang × **Regular** Lang → **Regular** Lang

Why Care About Closed Ops on Reg Langs?

- **Closed operations for regular langs preserve "regularness"**
- I.e., it **preserves the same computation model!**
- Can **"combine" smaller "regular" computations** to get **bigger ones:**

For Example:

U : **Regular** Lang × **Regular** Lang → **Regular** Lang

- In general, this semester, **we want operations that are closed!**

Is Union Closed For Regular Langs?

In this course, we are interested in closed operations for a set of languages (here the set of regular languages)

(In general, a set is **closed** under an operation if applying the operation to members of the set produces a result in the same set)

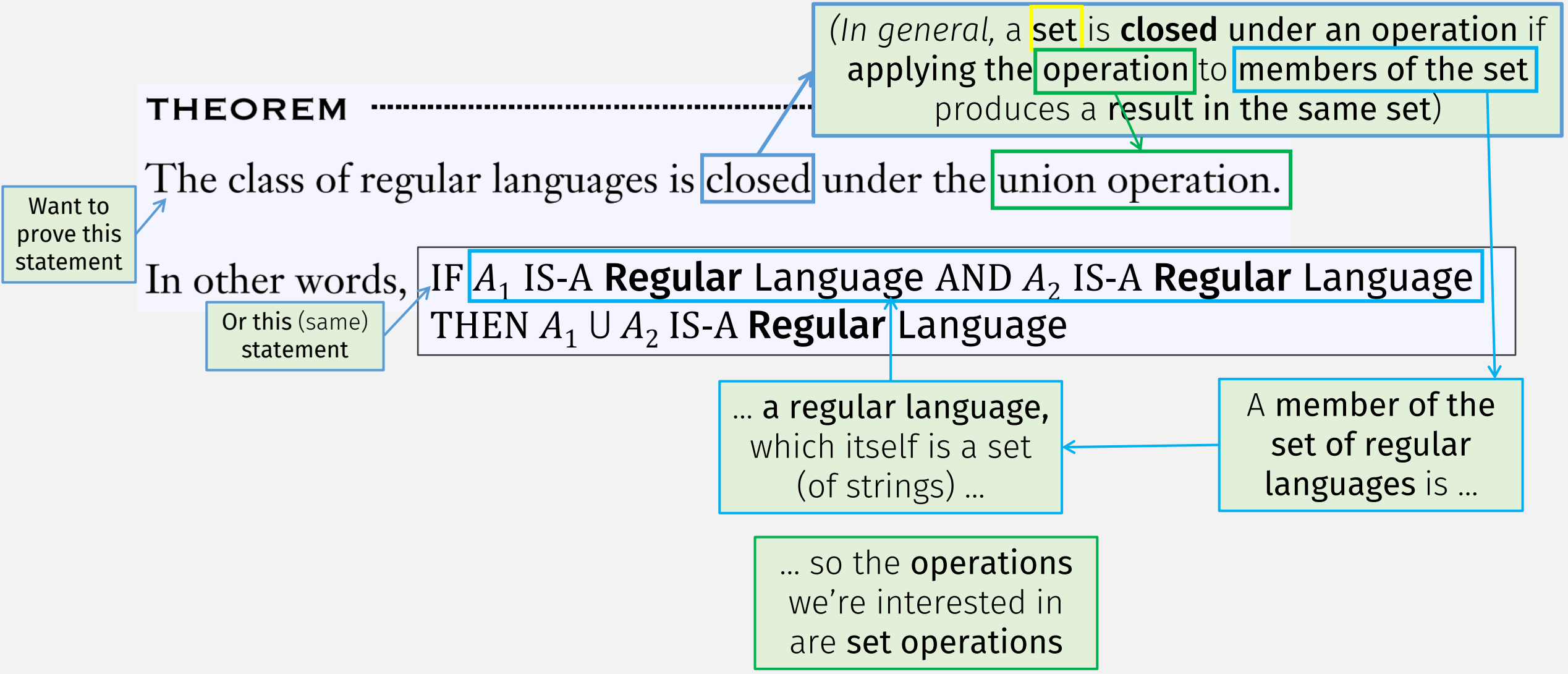
The class of regular languages is closed under the union operation.

Want to prove this statement

In other words, IF A_1 IS-A **Regular** Language AND A_2 IS-A **Regular** Language THEN $A_1 \cup A_2$ IS-A **Regular** Language

Or this (same) statement

Is Union Closed For Regular Langs?



Is Union Closed For Regular Langs?

THEOREM

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

In other words,

Want to prove this statement

IF A_1 IS-A **Regular** Language AND A_2 IS-A **Regular** Language THEN $A_1 \cup A_2$ IS-A **Regular** Language

Need to <u>prove</u> an IF-THEN statement!

Is Union Closed For Regular Langs?, i.e.,

IF A_1 IS-A **Regular** Language AND
A_2 IS-A **Regular** Language
THEN $A_1 \cup A_2$ IS-A **Regular** Language

`mk-IFTHEN-Proof` (with `ASSUMPTION` = proof of A_1 IS-A **Regular** Language AND A_2 IS-A **Regular** Language):

Statements	**Justifications**
1. A_1 IS-A **Regular** Language	**`1. FIRST ASSUMPTION`**
2. A_2 IS-A **Regular** Language	**`2. SECOND ASSUMPTION`**
3. M_1 IS-A **DFA** Machine AND M_1 **recognizes** A_1	**`3. APPLY`** DefRegLang **`TO`** Proof_1

???

DefRegLang

IF M IS-A **DFA** Machine AND M **recognizes** L
THEN L IS-A **Regular** Language

7. $A_1 \cup A_2$ IS-A **Regular** Language

IF A THEN B" == "IF B THEN A" ??

Statements

1. A_1 IS-A **Regular** Language
2. A_2 IS-A **Regular** Language
3. M_1 IS-A **DFA** Machine AND M_1 **recognizes** A_1

Justifications

1. **FIRST** ASSUMPTION
2. **SECOND** ASSUMPTION
3. **APPLY** DefRegLang **TO** Proof_1

???

DefRegLang

IF M IS-A **DFA** Machine AND M **recognizes** L
THEN L IS-A **Regular** Language

DefRegLang???

IF L IS-A **Regular** Language
THEN M IS-A **DFA** Machine AND M **recognizes** L

Equivalence of Conditional Statements

- Yes or No? **"If X then Y" is equivalent to:**
  - "If Y then X" (**converse**)
    - No!

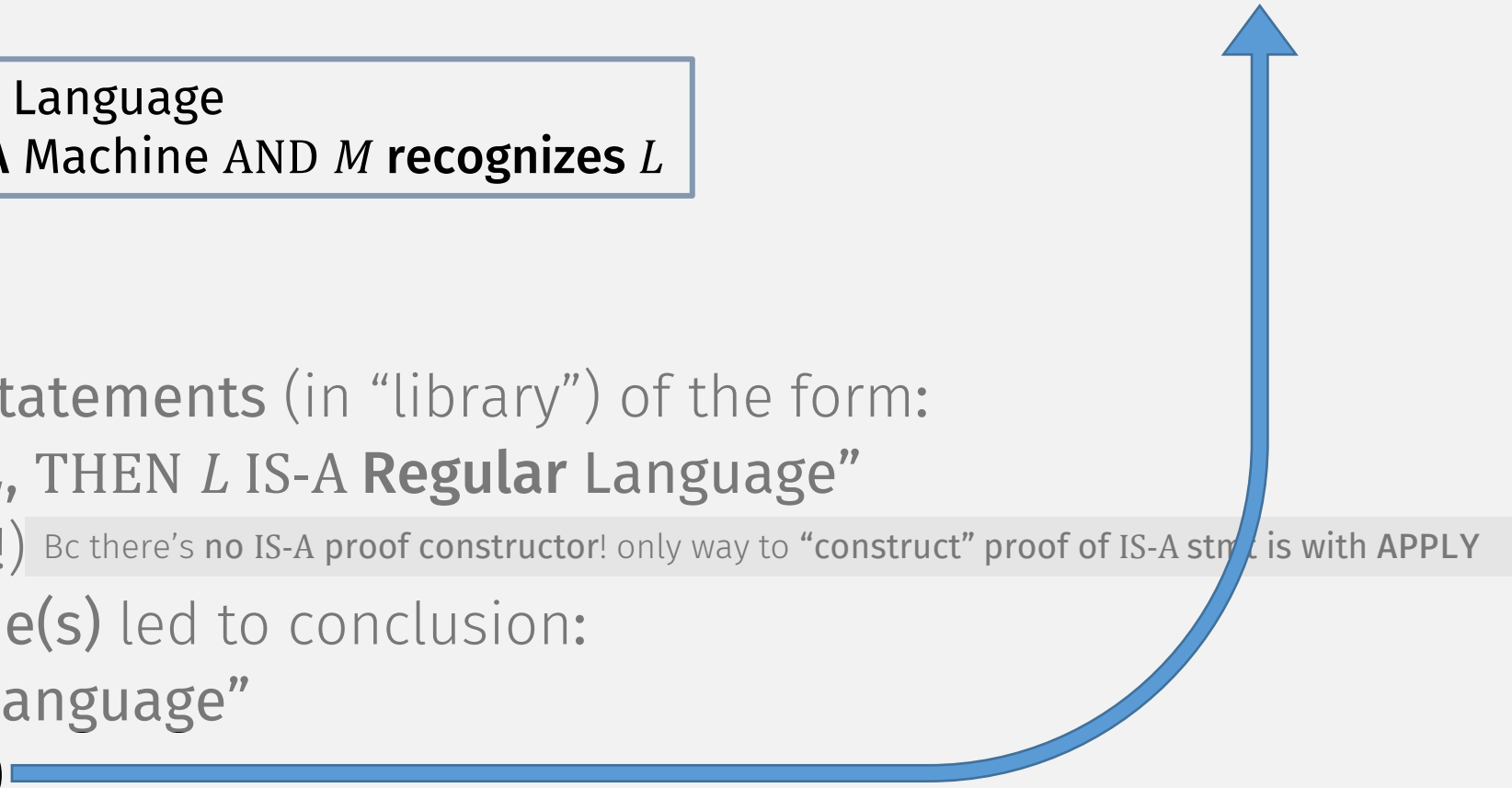
DefRegLang

IF M IS-A **DFA** Machine AND M **recognizes** L
THEN L IS-A **Regular** Language

If Regular, Then DFA?

- Prove: IF L IS-A **Regular** Language THEN M IS-A **DFA** Machine AND M **recognizes** L
- Proof (Sketch)

  Case analysis:
  - **Look at all** If-then **statements** (in "library") of the form:
    - "IF … language L, THEN L IS-A **Regular** Language"
  - (At least one is true!) Bc there's **no IS-A proof constructor**! only way to **"construct" proof of IS-A stmt is with APPLY**
  - **Figure out which one(s)** led to conclusion:
    - "L IS-A **Regular** Language"
  - (There's only 1 way!)



DefRegLangCoro

- So it must be that: IF L IS-A **Regular** Language THEN M IS-A **DFA** Machine AND M **recognizes** L

"Corollary"

(because there was only 1 possible way to prove stmt: L IS-A **Regular** Language)

Is Union Closed For Regular Langs?, i.e.,

IF A_1 IS-A **Regular** Language AND A_2 IS-A **Regular** Language
THEN $A_1 \cup A_2$ IS-A **Regular** Language

`mk-IFTHEN-Proof` (with `ASSUMPTION` = proof of A_1 IS-A **Regular** Language AND A_2 IS-A **Regular** Language):

Statements	**Justifications**
1. A_1 IS-A **Regular** Language	**`1. FIRST ASSUMPTION`**
2. A_2 IS-A **Regular** Language	**`2. SECOND ASSUMPTION`**
3. M_1 IS-A **DFA** Machine AND M_1 **recognizes** A_1	**`3. APPLY`** DefRegLang **`TO`** Proof_1

???

DefRegLang

IF M IS-A **DFA** Machine AND M **recognizes** L
THEN L IS-A **Regular** Language

DefRegLangCoro

IF L IS-A **Regular** Language
THEN M IS-A **DFA** Machine AND M **recognizes** L

7. $A_1 \cup A_2$ IS-A **Regular** Language

Is Union Closed For Regular Langs?, i.e.,

IF A_1 IS-A **Regular** Language AND
A_2 IS-A **Regular** Language
THEN $A_1 \cup A_2$ IS-A **Regular** Language

`mk-IFTHEN-Proof` (with `ASSUMPTION` = proof of A_1 IS-A **Regular** Language AND A_2 IS-A **Regular** Language):

Statements	**Justifications**
1. A_1 IS-A **Regular** Language	1. `FIRST ASSUMPTION`
2. A_2 IS-A **Regular** Language	2. `SECOND ASSUMPTION`
3. M_1 IS-A **DFA** Machine AND M_1 **recognizes** A_1	3. **APPLY** DefRegLangCoro **TO** Proof_1
4. M_2 IS-A **DFA** Machine AND M_2 **recognizes** A_2	4. **APPLY** DefRegLangCoro **TO** Proof_2
5. let M = (TODO: construct the DFA!) IS-A **DFA** Machine	????
6. M **recognizes** $A_1 \cup A_2$	????
7. $A_1 \cup A_2$ IS-A **Regular** Language	7.

DefRegLang

IF M IS-A **DFA** Machine AND M **recognizes** L
THEN L IS-A **Regular** Language

DefRegLangCoro

IF L IS-A **Regular** Language
THEN M IS-A **DFA** Machine AND M **recognizes** L

DEFINITION

A ***finite automaton*** is a `mk-DFA` $(Q, \Sigma, \delta, q_0, F)$, where

1. Q is a finite set called the ***states***,
2. Σ is a finite set called the ***alphabet***,
3. $\delta: Q \times \Sigma \longrightarrow Q$ is the ***transition function***,
4. $q_0 \in Q$ is the ***start state***, and
5. $F \subseteq Q$ is the ***set of accept states***.

3. M_1 IS-A **DFA Machine** AND M_1 **recognizes** A_1
4. M_2 IS-A **DFA Machine** AND M_2 **recognizes** A_2
5. let M = (TODO: construct the DFA!) IS-A **DFA Machine**

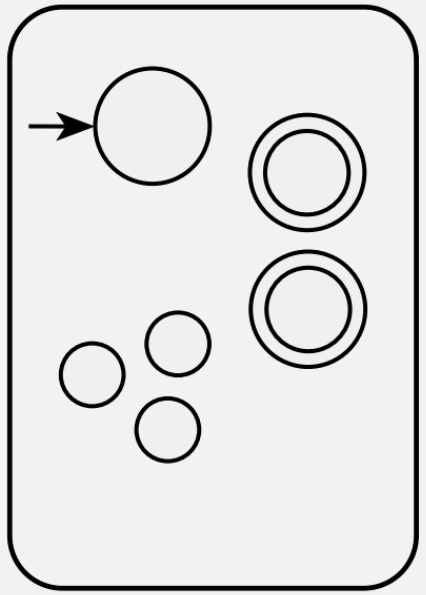
Problem: We **don't know** exactly what M_1 and M_2 are... but we **still know**...

$M_1 =$ **`mk-DFA`**$(Q_1, \Sigma, \delta_1, q_1, F_1)$ for some Q_1, etc ...

$M_2 =$ **`mk-DFA`**$(Q_2, \Sigma, \delta_2, q_2, F_2)$ for some Q_2, etc ...

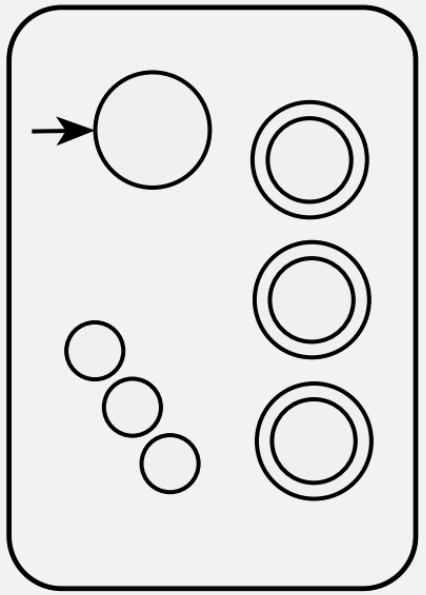
M_1

recognizes A_1



M_2

recognizes A_2



DEFINITION

A ***finite automaton*** is a `mk-DFA` $(Q, \Sigma, \delta, q_0, F)$, where

1. Q is a finite set called the ***states***,
2. Σ is a finite set called the ***alphabet***,
3. $\delta: Q \times \Sigma \longrightarrow Q$ is the ***transition function***,
4. $q_0 \in Q$ is the ***start state***, and
5. $F \subseteq Q$ is the ***set of accept states***.

Problem: We **don't know** exactly what M_1 and M_2 are... but we **still know**...

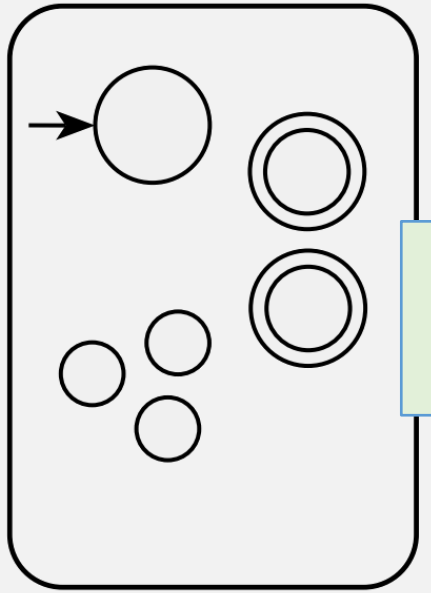
M_1 = **mk-DFA**(Q_1, Σ, δ_1, q_1, F_1) for some Q_1, etc ...

M_2 = **mk-DFA**(Q_2, Σ, δ_2, q_2, F_2) for some Q_2, etc ...

Union

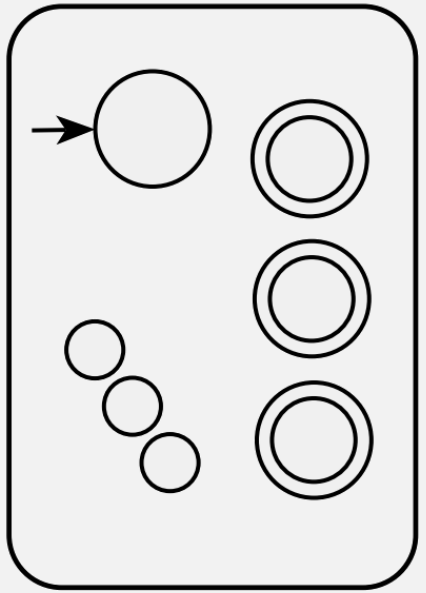
M_1

recognizes A_1



M_2

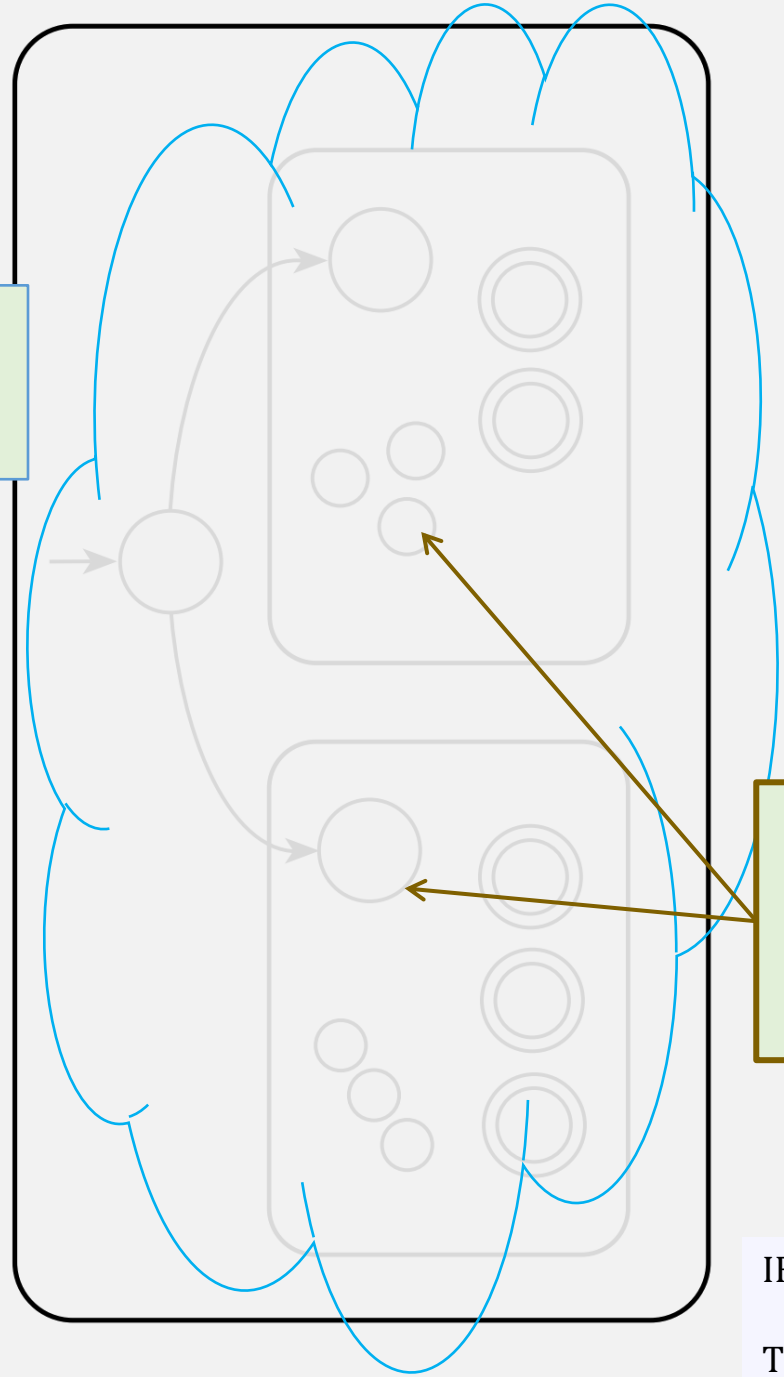
recognizes A_2



Want: M

Recognizes $A_1 \cup A_2$

(to prove $A_1 \cup A_2$ IS-A **Regular** Language)



Rough sketch Idea: M is a combination of M_1 and M_2 that: checks whether its input is accepted by either M_1 or M_2

But: a DFA can only read its input once!

Need to: somehow simulate "being in" both an M_1 and M_2 state simultaneously

IF A_1 IS-A **Regular** Language AND
A_2 IS-A **Regular** Language
THEN $A_1 \cup A_2$ IS-A **Regular** Language

Is Union Closed For Regular Langs?

Proof (continuation)

Define function $\textbf{UNION}_{\textbf{DFA}} : \textbf{DFA} \times \textbf{DFA} \rightarrow \textbf{DFA}$

$\textbf{UNION}_{\textbf{DFA}}$ $(M_1, M_2) = M$, where

- Input: $M_1 = \textbf{mk-DFA}(\ Q_1, \Sigma, \delta_1, q_1, F_1\ )$
  $M_2 = \textbf{mk-DFA}(\ Q_2, \Sigma, \delta_2, q_2, F_2\ )$
- Construct: $M = \textbf{mk-DFA}(\ Q, \Sigma, \delta, q_0, F\ )$ using M_1, M_2 parts (that accepts if either accepts)
- states of M:

$$Q = \{(r_1, r_2) \mid r_1 \in Q_1 \text{ and } r_2 \in Q_2\} \quad = Q_1 \times Q_2$$

This set is the ***Cartesian product*** of sets Q_1 and Q_2

Want: M that can simultaneously "be in" both an M_1 and M_2 state

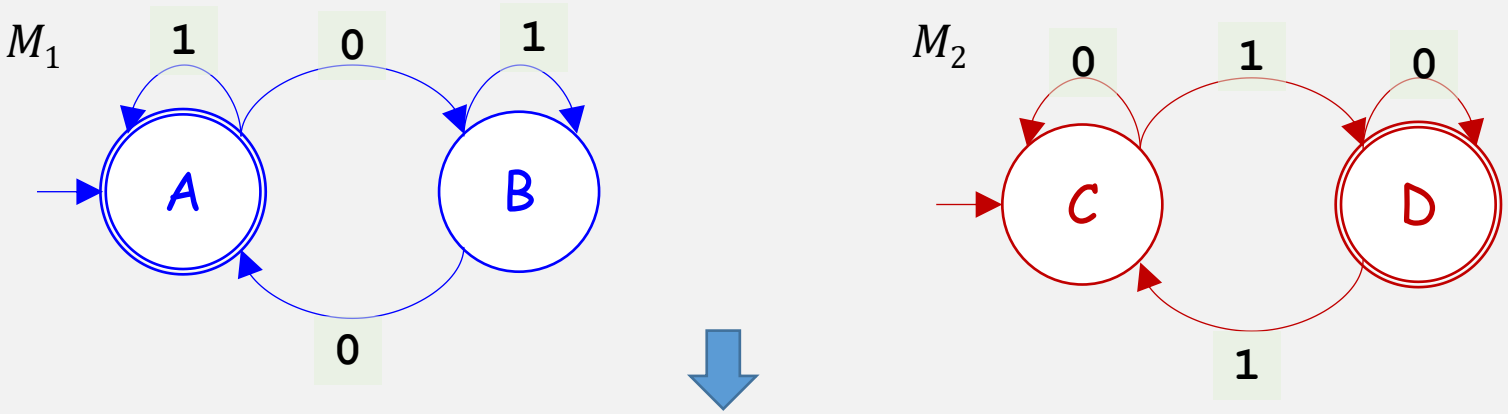
A state of M is a pair:
- first part: state of M_1
- second part: state of M_2

states of M: all possible pair combinations of states of M_1 and M_2

A ***finite automaton*** is a mk-DFA $(Q, \Sigma, \delta, q_0, F)$, where

1. Q is a finite set called the ***states***,
2. Σ is a finite set called the ***alphabet***,
3. $\delta : Q \times \Sigma \longrightarrow Q$ is the ***transition function***,[1]
4. $q_0 \in Q$ is the ***start state***, and
5. $F \subseteq Q$ is the ***set of accept states***.

DFA Union Example



https://cs.wellesley.edu/~cs235/fall10/lectures/14_DFA_operations_revised_2.pdf

Is Union Closed For Regular Langs?

Proof (continuation)

Define function $\mathbf{UNION_{DFA} : DFA \times DFA \rightarrow DFA}$

$\mathbf{UNION_{DFA}}$ $(M_1, M_2) = M$, where

- Input: $M_1 = \texttt{mk-DFA}(\ Q_1,\ \Sigma,\ \delta_1,\ q_1,\ F_1\ )$
  $M_2 = \texttt{mk-DFA}(\ Q_2,\ \Sigma,\ \delta_2,\ q_2,\ F_2\ )$
- Construct: $M = \texttt{mk-DFA}(\ Q,\ \Sigma,\ \delta,\ q_0,\ F\ )$ **using M_1, M_2 parts** (that accepts if either accepts)
- states of M:

$$Q = \{(r_1, r_2) \mid r_1 \in Q_1 \text{ and } r_2 \in Q_2\} \quad = Q_1 \times Q_2$$

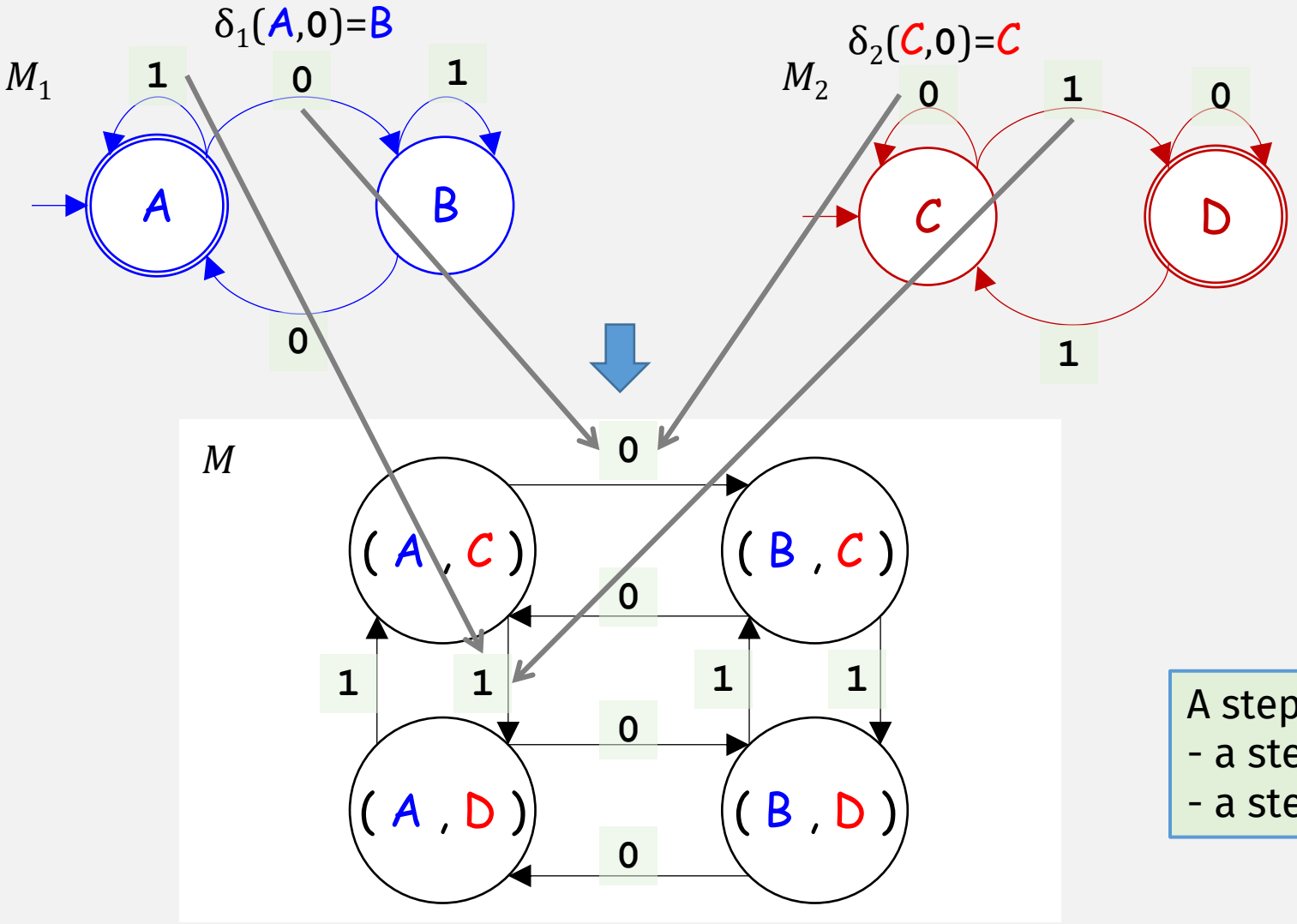
This set is the ***Cartesian product*** of sets Q_1 and Q_2

$$\delta((r_1, r_2), a) = (\delta_1(r_1, a), \delta_2(r_2, a))$$

A step in M is <u>both</u>:
- a step in M_1, and
- a step in M_2

A ***finite automaton*** is a `mk-DFA` $(Q, \Sigma, \delta, q_0, F)$, where

1. Q is a finite set called the ***states***,
2. Σ is a finite set called the ***alphabet***,
3. $\delta : Q \times \Sigma \longrightarrow Q$ is the ***transition function***,
4. $q_0 \in Q$ is the ***start state***, and
5. $F \subseteq Q$ is the ***set of accept states***.



A step in M is <u>both</u>:
- a step in M_1, and
- a step in M_2

Is Union Closed For Regular Langs?

Proof (continuation)

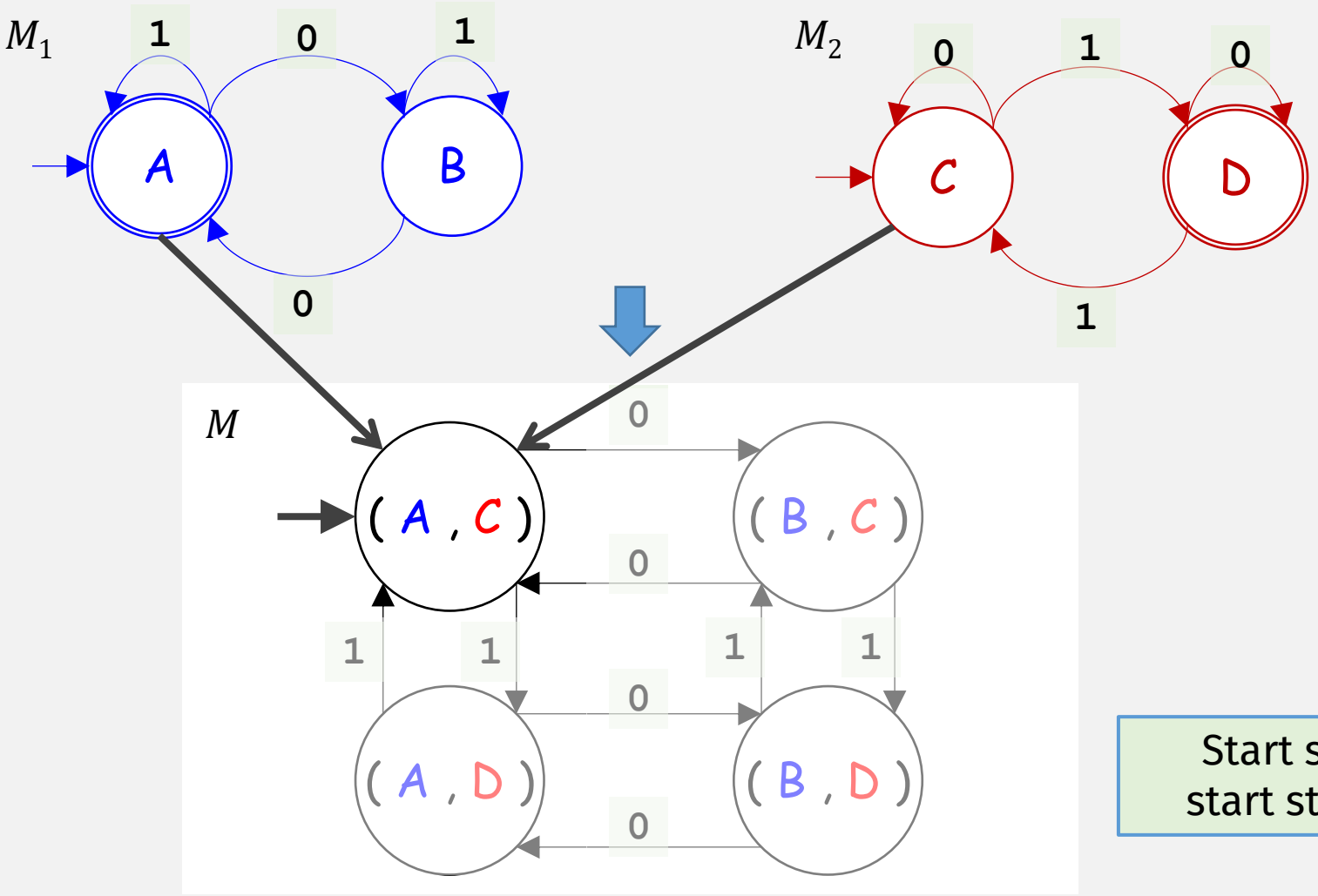
Define function $\mathbf{UNION_{DFA} : DFA \times DFA \rightarrow DFA}$

$\mathbf{UNION_{DFA}}\ (M_1, M_2) = M$, where

- Input: $M_1 = \mathbf{mk\text{-}DFA}(\ Q_1,\ \Sigma,\ \delta_1,\ q_1,\ F_1\ )$
  $M_2 = \mathbf{mk\text{-}DFA}(\ Q_2,\ \Sigma,\ \delta_2,\ q_2,\ F_2\ )$
- Construct: $M = \mathbf{mk\text{-}DFA}(\ Q,\ \Sigma,\ \delta,\ q_0,\ F\ )$ **using M_1, M_2 parts** (that accepts if either accepts)
- states of M: $Q = \{(r_1, r_2) \mid r_1 \in Q_1 \text{ and } r_2 \in Q_2\} = Q_1 \times Q_2$
  This set is the ***Cartesian product*** of sets Q_1 and Q_2
- M transition fn: $\delta((r_1, r_2), a) = (\delta_1(r_1, a), \delta_2(r_2, a))$
- **M start state:** (q_1, q_2)

Start state of M is <u>both</u> start states of M_1 and M_2

DFA Union Example



Start state of M is <u>both</u> start states of M_1 and M_2

https://cs.wellesley.edu/~cs235/fall10/lectures/14_DFA_operations_revised_2.pdf

Is Union Closed For Regular Langs?

Proof (continuation)

Define function $\textbf{UNION}_{\textbf{DFA}} : \textbf{DFA} \times \textbf{DFA} \rightarrow \textbf{DFA}$

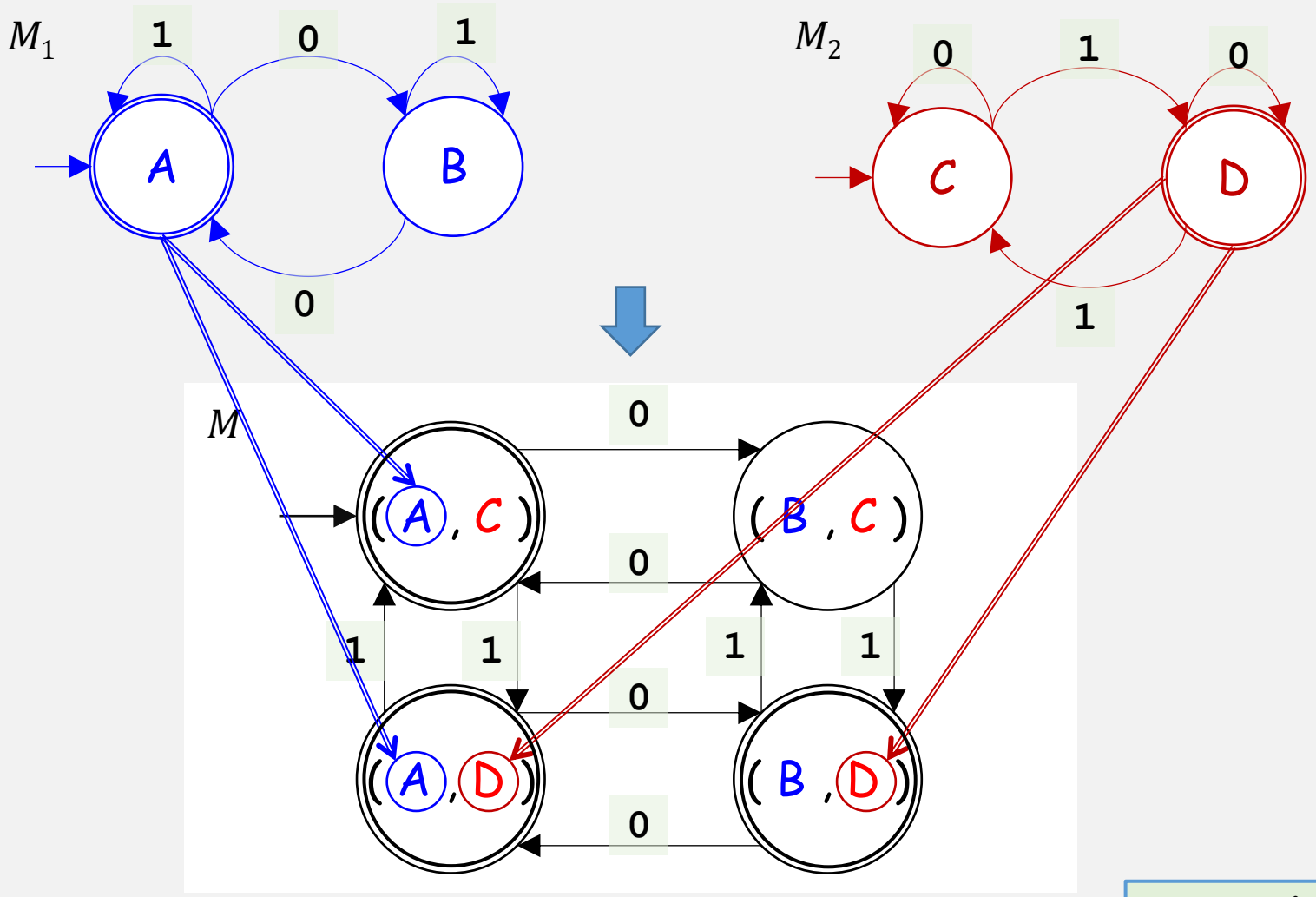
$\textbf{UNION}_{\textbf{DFA}}\ (M_1, M_2) = M$, where

- Input: $M_1 = \textbf{mk-DFA}(\ Q_1,\ \Sigma,\ \delta_1,\ q_1,\ F_1\ )$
  $M_2 = \textbf{mk-DFA}(\ Q_2,\ \Sigma,\ \delta_2,\ q_2,\ F_2\ )$
- Construct: $M = \textbf{mk-DFA}(\ Q,\ \Sigma,\ \delta,\ q_0,\ F\ )$ **using M_1, M_2 parts** (that accepts if either accepts)
- states of M: $Q = \{(r_1, r_2) |\ r_1 \in Q_1 \text{ and } r_2 \in Q_2\}\ = Q_1 \times Q_2$
  This set is the ***Cartesian product*** of sets Q_1 and Q_2
- M transition fn: $\delta((r_1, r_2), a) = (\delta_1(r_1, a), \delta_2(r_2, a))$
- M start state: (q_1, q_2)
- M accept states: $F = \{(r_1, r_2) |\ r_1 \in F_1 \text{ or } r_2 \in F_2\}$

Remember:
Accept states must be subset of Q

Accept if <u>either</u> M_1 or M_2 accept

DFA Union Example



Accept if either M_1 or M_2 accept

https://cs.wellesley.edu/~cs235/fall10/lectures/14_DFA_operations_revised_2.pdf

Is Union Closed For Regular Langs?

Proof (continuation)

Define function $\textbf{UNION}_{\textbf{DFA}} : \textbf{DFA} \times \textbf{DFA} \rightarrow \textbf{DFA}$

$\textbf{UNION}_{\textbf{DFA}}\ (M_1, M_2) = M$, where

- Input: $M_1 = \textbf{mk-DFA}(\ Q_1,\ \Sigma,\ \delta_1,\ q_1,\ F_1\ )$
  $M_2 = \textbf{mk-DFA}(\ Q_2,\ \Sigma,\ \delta_2,\ q_2,\ F_2\ )$
- Construct: $M = \textbf{mk-DFA}(\ Q,\ \Sigma,\ \delta,\ q_0,\ F\ )$ **using M_1, M_2 parts** (that accepts if either accepts)
- **states of M:** $Q = \{(r_1, r_2) |\ r_1 \in Q_1 \text{ and } r_2 \in Q_2\}$ $= Q_1 \times Q_2$
  This set is the ***Cartesian product*** of sets Q_1 and Q_2
- **M transition fn:** $\delta((r_1, r_2), a) = (\delta_1(r_1, a), \delta_2(r_2, a))$
- **M start state:** (q_1, q_2)
- **M accept states:** $F = \{(r_1, r_2) |\ r_1 \in F_1 \text{ or } r_2 \in F_2\}$

Q.E.D.? 

Is Union Closed For Regular Langs?, i.e.,

IF A_1 IS-A **Regular** Language AND
A_2 IS-A **Regular** Language
THEN $A_1 \cup A_2$ IS-A **Regular** Language

`mk-IFTHEN-Proof` (with `ASSUMPTION` = **proof of** A_1 IS-A **Regular** Language AND A_2 IS-A **Regular** Language):

Statements	**Justifications**
1. A_1 IS-A **Regular** Language	1. **`FIRST`** `ASSUMPTION`
2. A_2 IS-A **Regular** Language	2. **`SECOND`** `ASSUMPTION`
3. M_1 IS-A **DFA** Machine AND M_1 **recognizes** A_1	3. **APPLY** DefRegLangCoro **TO** Proof_1
4. M_2 IS-A **DFA** Machine AND M_2 **recognizes** A_2	4. **APPLY** DefRegLangCoro **TO** Proof_2
5. let M = (TODO: construct the DFA!) IS-A **DFA** Machine	????
6. M **recognizes** $A_1 \cup A_2$	????
7. $A_1 \cup A_2$ IS-A **Regular** Language	7. **APPLY** DefRegLang **TO** **`mk-AND-proof`** Proof_5, Proof_6

Is Union Closed For Regular Langs?, i.e.,

IF A_1 IS-A **Regular** Language AND
A_2 IS-A **Regular** Language
THEN $A_1 \cup A_2$ IS-A **Regular** Language

`mk-IFTHEN-Proof` (with `ASSUMPTION` = proof of A_1 IS-A **Regular** Language AND A_2 IS-A **Regular** Language):

Statements	**Justifications**
1. A_1 IS-A **Regular** Language	1. **FIRST** ASSUMPTION
2. A_2 IS-A **Regular** Language	2. **SECOND** ASSUMPTION
3. M_1 IS-A **DFA** Machine AND M_1 **recognizes** A_1	3. **APPLY** DefRegLangCoro **TO** Proof_1
4. M_2 IS-A **DFA** Machine AND M_2 **recognizes** A_2	4. **APPLY** DefRegLangCoro **TO** Proof_2
5. let $M = \text{UNION}_{\text{DFA}}(M_1, M_2)$ IS-A **DFA** Machine	**5.** **APPLY** $\text{UNION}_{\text{DFA}} : \text{DFA} \times \text{DFA} \to \text{DFA}$ **TO** $M_1 : \text{DFA}, M_2 : \text{DFA}$????
6. M **recognizes** $A_1 \cup A_2$	????
7. $A_1 \cup A_2$ IS-A **Regular** Language	**7. APPLY** DefRegLang **TO** **`mk-AND-proof`** $\text{Proof}_5, \text{Proof}_6$

Is Union Closed For Regular Langs?, i.e.,

IF A_1 IS-A **Regular** Language AND
A_2 IS-A **Regular** Language
THEN $A_1 \cup A_2$ IS-A **Regular** Language

`mk-IFTHEN-Proof` (with `ASSUMPTION` = proof of A_1 IS-A **Regular** Language AND A_2 IS-A **Regular** Language):

Statements	**Justifications**
1. A_1 IS-A **Regular** Language	1. `FIRST ASSUMPTION`
2. A_2 IS-A **Regular** Language	2. `SECOND ASSUMPTION`
3. M_1 IS-A **DFA** Machine AND M_1 **recognizes** A_1	3. **APPLY** DefRegLangCoro **TO** Proof_1
4. M_2 IS-A **DFA** Machine AND M_2 **recognizes** A_2	4. **APPLY** DefRegLangCoro **TO** Proof_2
5. let $M = \text{UNION}_{\text{DFA}}(M_1, M_2)$ IS-A **DFA** Machine	**5. APPLY** $\text{UNION}_{\text{DFA}} : \textbf{DFA} \times \textbf{DFA} \rightarrow \textbf{DFA}$ **TO** $M_1 : \textbf{DFA}, M_2 : \textbf{DFA}$
6. M **recognizes** $A_1 \cup A_2$	**6. BY** (TODO: create Example Table!) **COMPARING** $M, A_1 \cup A_2$
7. $A_1 \cup A_2$ IS-A **Regular** Language	**7. APPLY** DefRegLang **TO** `mk-AND-proof` $\text{Proof}_5, \text{Proof}_6$