Module 02
Models of Computation
Finite State Machines

CS 332 Organization of Programming Languages
Embry-Riddle Aeronautical University
Daytona Beach, FL

M02 Outcomes

At the end of this module you should be able to …

1. Given a grammar, justify why it is or is not regular.
2. State the definition of the Finite State Machine (FSM) and its component parts.
3. Given a regular language create the Finite State Machine for it.
4. State the acceptance criteria for an FSM.
5. Given an FSM and a string, demonstrate the state transitions for processing the string.
6. Given an FSM and a string, state whether the FSM accepts or rejects the string.
7. Provide real world examples of an FSM with justification.

Finite State Machines: Introduction (1/2)

- Finite State Machines (FSM) have theoretical value.
- Computer Science asks “What can a computer compute?”
- To answer, we need a formal (mathematical) model of what computation is.
- Pure computation is the mindless manipulation of symbols using formal rules:
  - Mindless – or a computer couldn’t do it!
  - Manipulation of symbols – operators modifying operands
  - Formal rules – require no judgment, opinion, thought. Just do what is says.
- FSMs are <u>one</u> of many models of computation.
- FSMs are a weak model of computation – they have no memory.

Finite State Machines: Introduction (2/2)

- FSMs have practical value

- FSMs are still used everywhere – all systems have state
  - Your car has state: on/off, locked/unlocked, in park/reverse/drive, etc.
  - Aircraft have state: at gate, taxi-ing, wheels up, wheels down, etc.
  - Soda machines have state: amount of money put in, amount of soda, etc
  - Video game non-player characters typically built on state machines.

- CS 332 consider FSMs as <u>language recognizers</u>.
  - Typical notion of FSM in formal language theory.
  - FSMs will <u>*accept*</u> anything in the language, and <u>*reject*</u> anything not in language.
  - Each language has it's own FSM (actually many FSM's).

M02 Language Definition (Review)

- A symbol is a single, distinct mark or character used as a representation.
- An alphabet, Σ, is a finite set of symbols.
- A string, u, is a sequence of symbols from some Σ.
- A language, L, is a (potentially infinite) set of strings.
  - Languages have no inherent meaning.
  - Languages are not about communication. Some are used for communication.
  - Languages are not about programming. Some are used for programming.

M02 Grammar Definition (Review)

- A grammar, G, supplies the rules by which a language is constructed
  - We're assuming there is a pattern or structure to the language
  - A set of random strings is still a language
  - Therefore, grammars make sense only for non-random languages

- Backus-Naur Form (BNF) is widely used, universally in Comp Sci
  - BNF uses <u>production rules</u> composed of a left hand side (LHS), and a right hand side (RHS), each containing symbols.
  - The symbols on left hand side (LHS) can be replaced by those on the right hand side (RHS).
  - Production rules take the form LHS → RHS (some authors use LHS := RHS).
  - Example: If you have a rule that says A → aCa, and the string abAca, you can use the rule to replace the "A" in the string with "aCa" – result is ab<u>aCa</u>ca (underlined for visibility).

M02 The Chomsky Hierarchy (Review)

- For now, define the Chomsky Hierarchy in terms of grammars
- Will describe restrictions on the Left Hand Side (LHS) and Right Hand Side (RHS)
- “Unrestricted” is sometimes called “recursively enumerable” but not covering that in this course – it would take several weeks.

Type	Name	Characteristics of Grammar
Type 3	Regular	LHS must contain exactly one non-terminal. Number of terminals in RHS cannot decrease. Strings derived from right to left, or left to right.
Type 2	Context Free	LHS must contain exactly one non-terminal. Number of terminals in RHS cannot decrease.
Type 1	Context Sensitive	LHS may have terminals and non-terminals. Number of terminals in RHS cannot decrease.
Type 0	Unrestricted	No restrictions

Models of Computation and the Chomsky Hierarchy

- Previously defined the Chomsky Hierarchy in terms of grammars
- Can also define in terms of model of computation
- Will discuss why these models and language match as we go through

Type	Name	Equivalent Model of Computation
Type 3	Regular	Finite State Machine, FSM, also known as Discrete Finite Automata, DFA
Type 2	Context Free	Stack Machine (Push Down Automata, PDA)
Type 1	Context Sensitive	Turing Machine, TM, with finite tape
Type 0	Unrestricted	Turing Machine, TM, with infinite tape

Finite State Machines (FSM): Definitions

- A Finite State Machine (FSM) is a mathematical device to recognize certain languages.
    - An FSM will *accept* all strings in the language it is designed for.
    - An FSM will *reject* all strings not in the language it is designed for.
    - Each language may have many FSMs; most are trivial.
    - The class of language FSMs recognize are called the *regular languages*.

FSM: Formal Definition

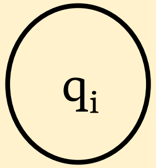
- Formally, a FSM $M = \{Q, \Sigma, q_o, F, \delta\}$, where,
  - Q = a finite set of states, (hence FINITE STATE Machine)
  - Σ = a finite set of symbols
  - q_o = a single *start state*, where processing begins ($q_o \in Q$)
  - F = a set of *final*, or *accepting*, states. (F subset of Q)
  - δ = a transition function

FSM Operation

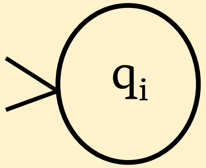
- A FSM, M, processes a string, u, as follows:
  - M begins in the start state, q_o
  - M processes string u one symbol at a time, from left to right.
  - Each symbol that is processed results in a transition to a new state according to the transition function, δ.
  - Once all symbols in u have been processed, M will ***accept*** the string if M is in a final state, and will ***reject*** the string otherwise. (This is why final states are also called accepting states.)
  - Accepted strings are in the language, L, and rejected strings are not

FSM Graphical Representation

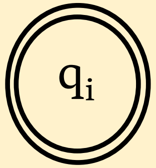
State (A circle with state name in it):



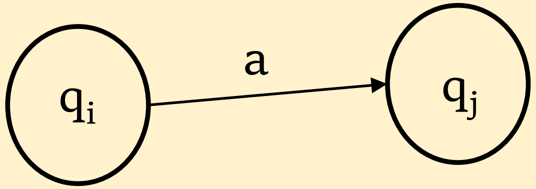
Start state (a circle with a duck beak):



Final state (double circle):



Transition (arrow labelled with symbol causing it):

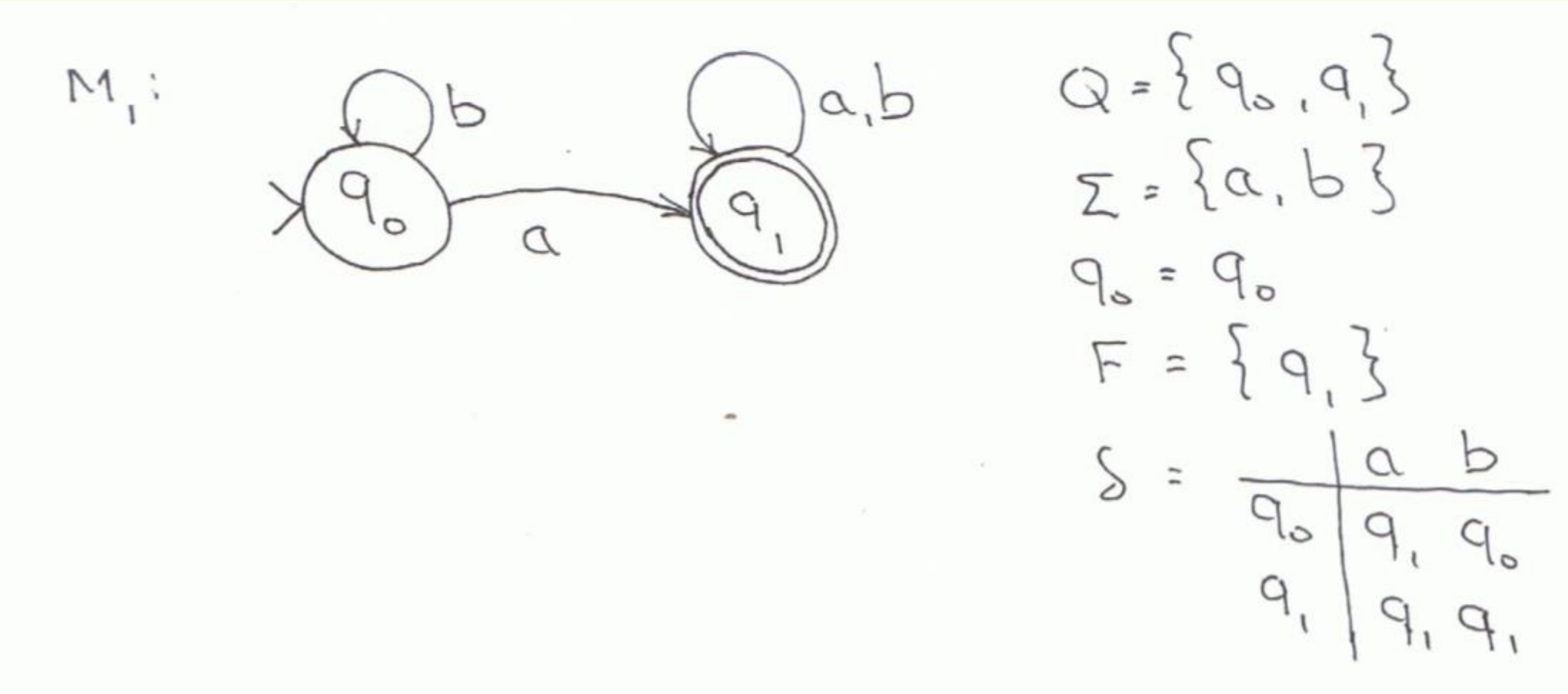


FSM Example #1

- Let $\Sigma = \{a, b\}$ (For all examples unless stated otherwise)
- Let L_1 = All strings having 1 or more ‘a.’

FSM Example #1

- Let L_1 = All strings having 1 or more ‘a.’

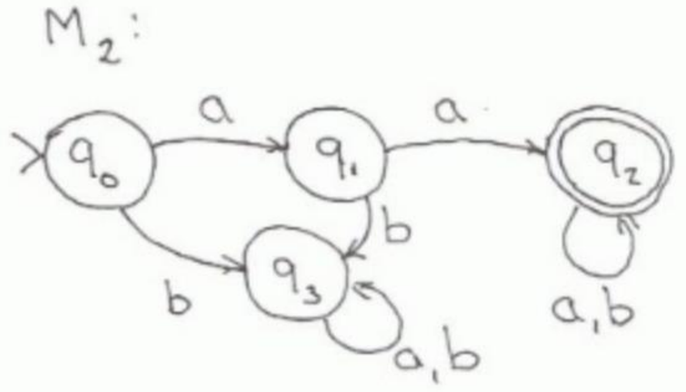


FSM Example #2

- Let L_2 = All strings that start with two ‘a’s.

FSM Example #2

- Let L_2 = All strings that start with two 'a's.



$Q = \{q_0, q_1, q_2, q_3\}$

$\Sigma = \{a, b\}$

$q_0 = q_0$

$F = \{q_2\}$

$\delta =$

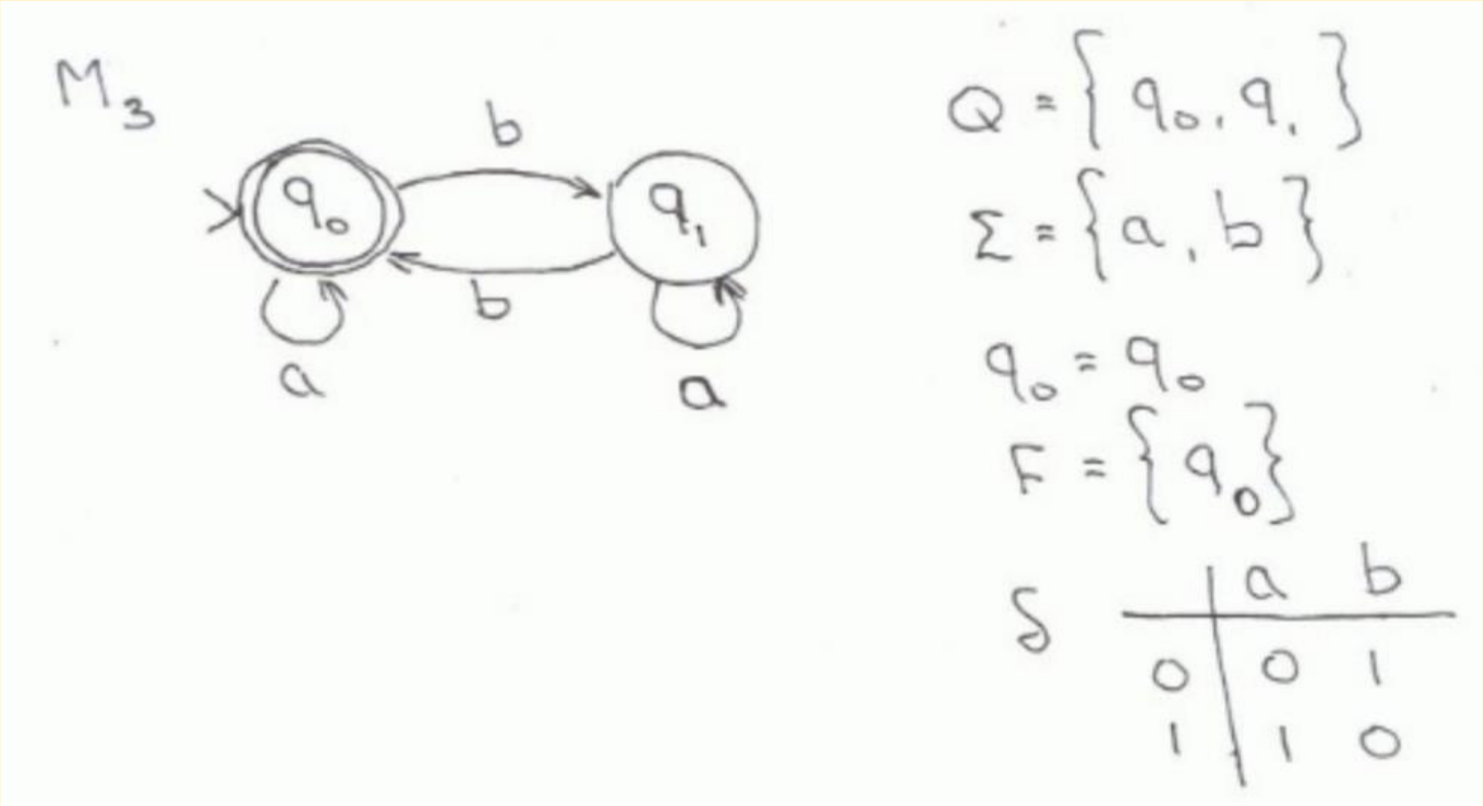
	a	b
0	1	3
1	2	3
2	2	2
3	3	3

FSM Example #3

- Let L_3 = All strings that have an even number of ‘b’s (including λ).

FSM Example #3

- Let L_3 = All strings that have an even number of ‘b’s (including λ).

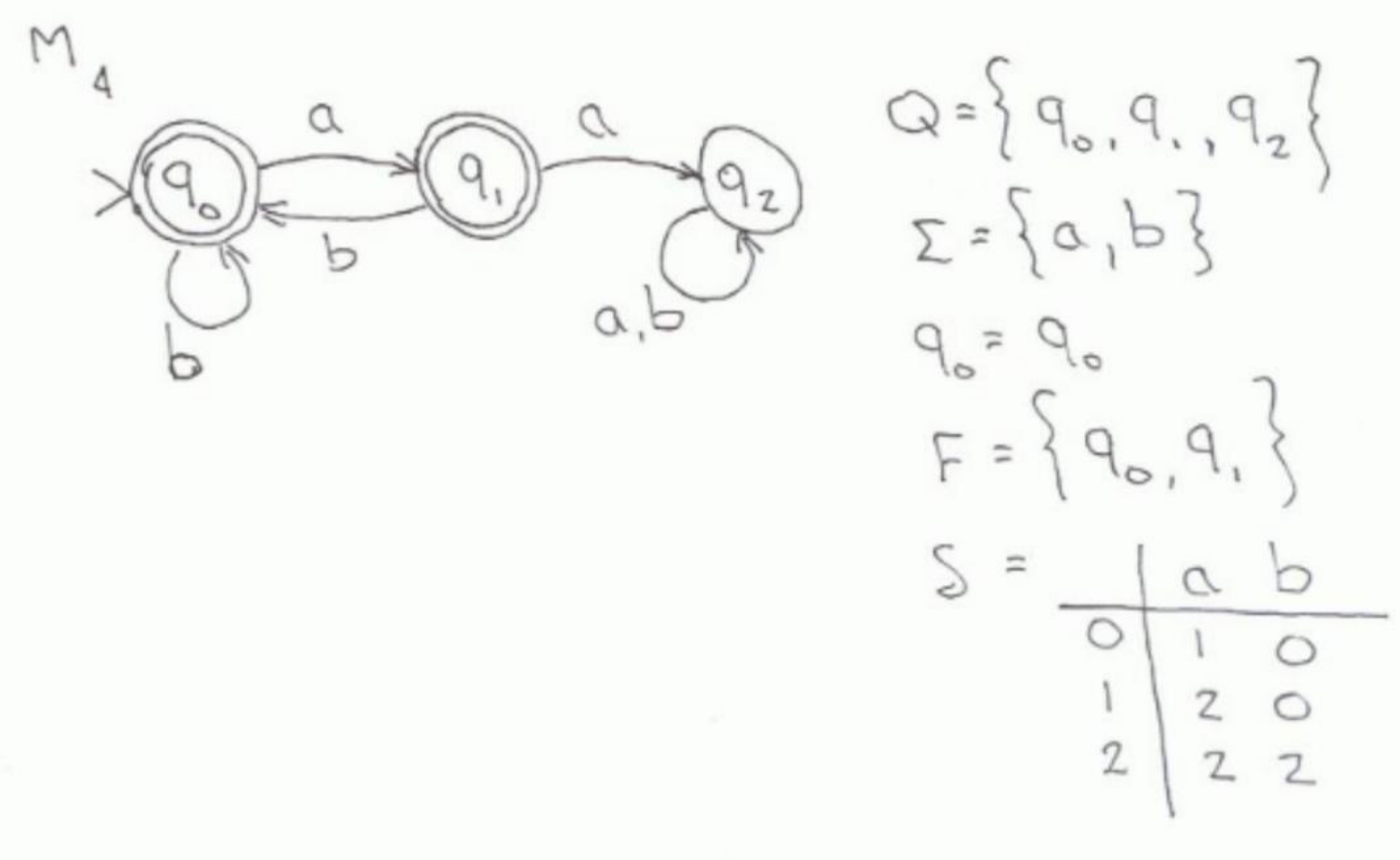


FSM Example #4

- Let L_4 = All strings where no two ‘a’s are adjacent, including λ.

FSM Example #4

- Let L_4 = All strings where no two ‘a’s are adjacent, including λ.

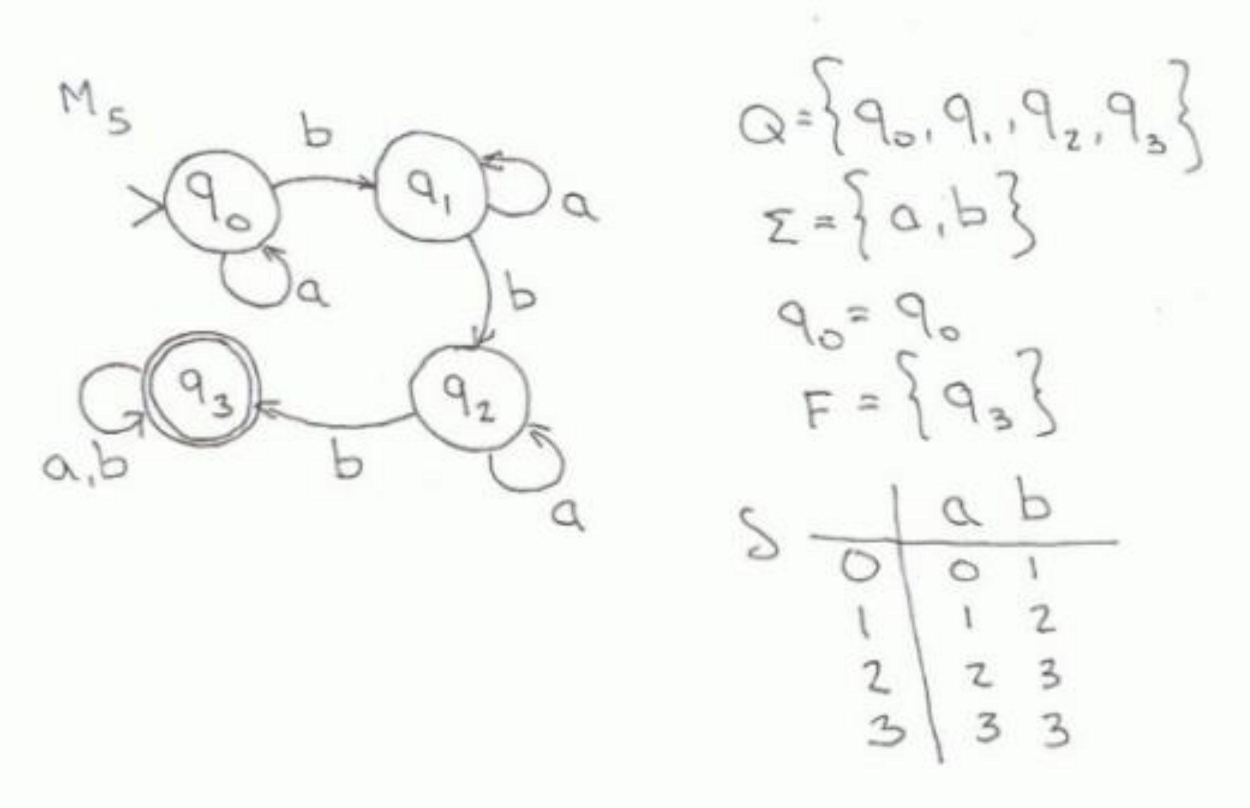


FSM Example #5

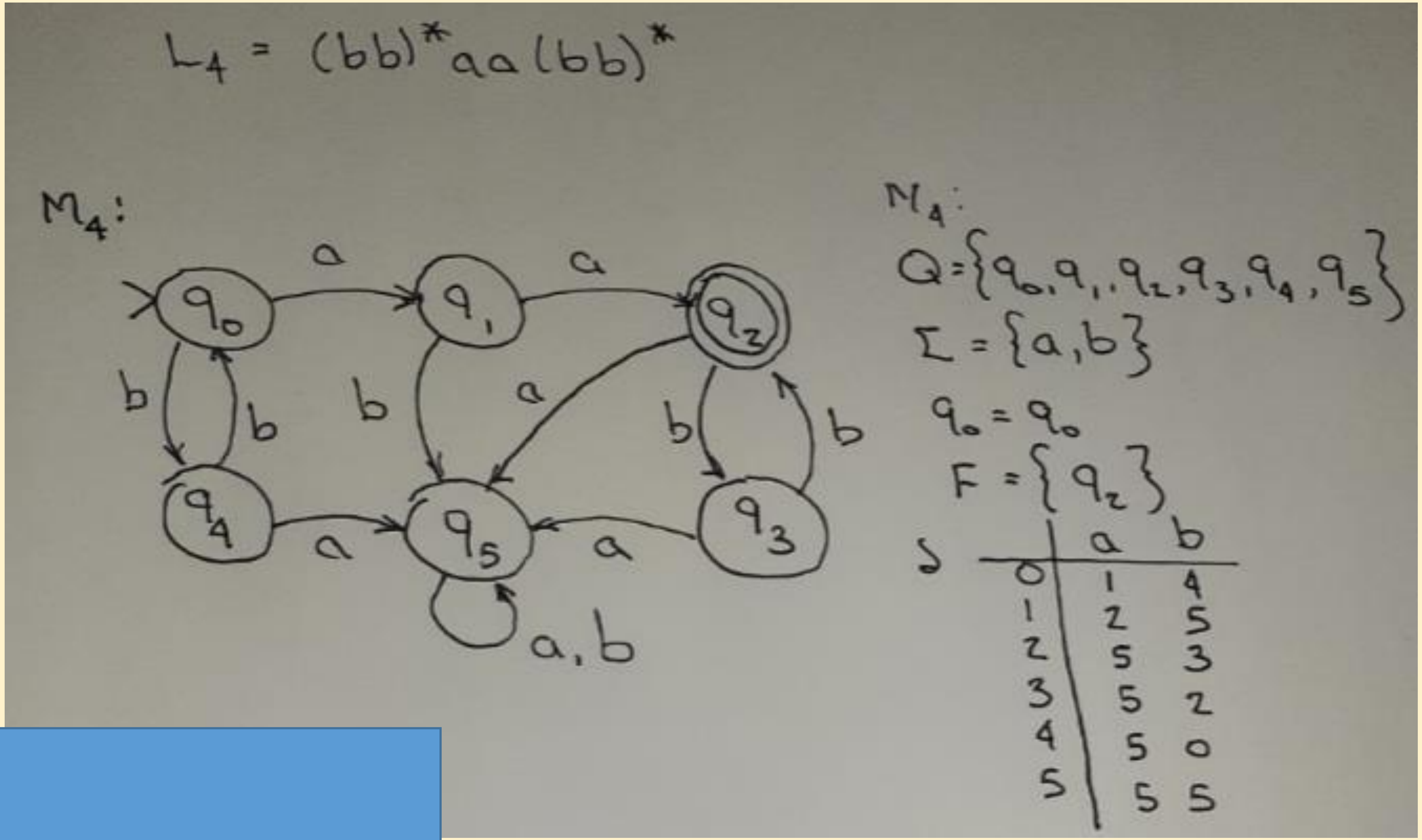
- Let L_5 = All strings with at least three ‘b’s.

FSM Example #5

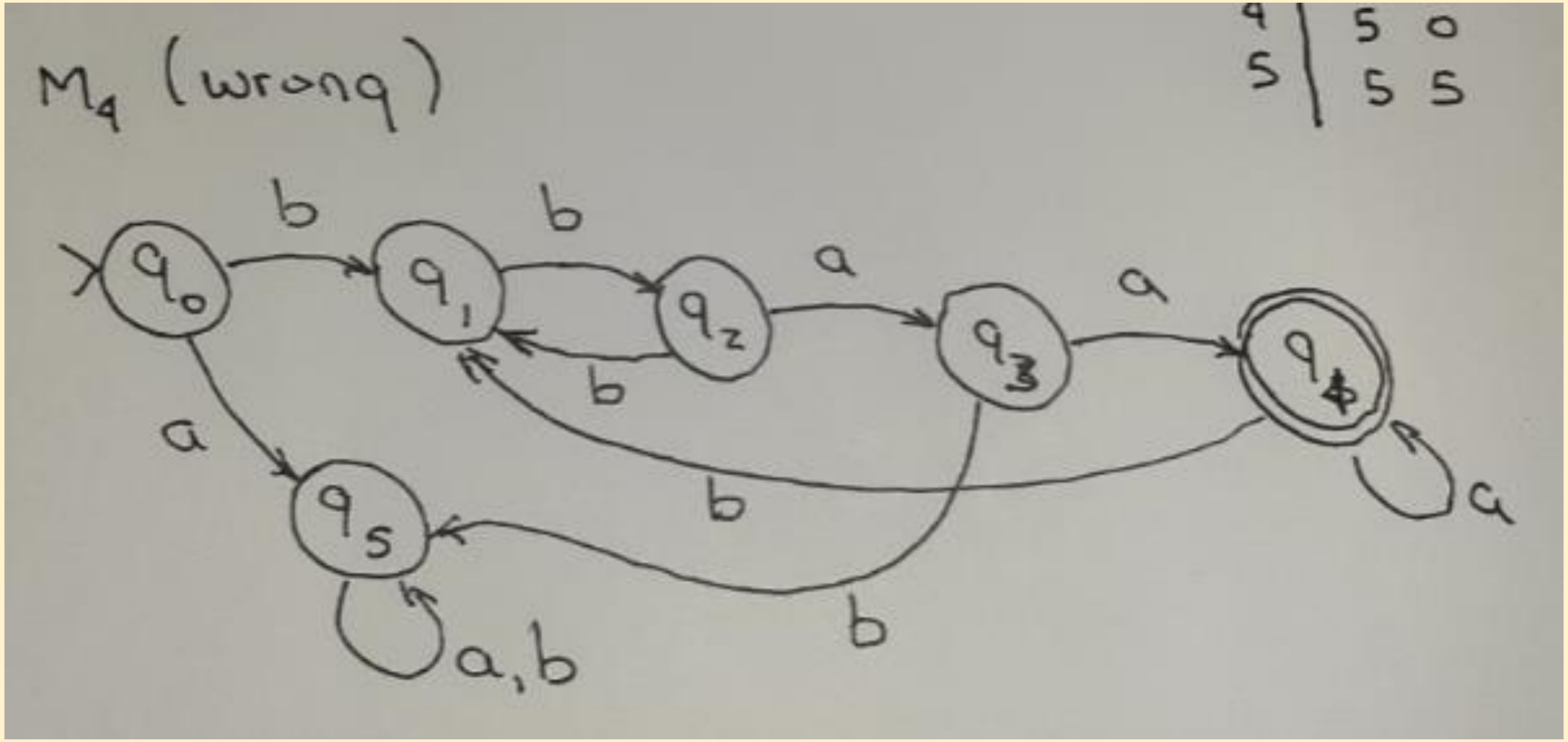
- Let L_5 = All strings with at least three ‘b’s.



- Let Σ = {a, b}, and L = (bb)*aa(bb)*
- Draw the FSM for L and provide its M = {Q, Σ, q_o, F, δ} form.
  - We'll also draw an incorrect FSM and discuss.



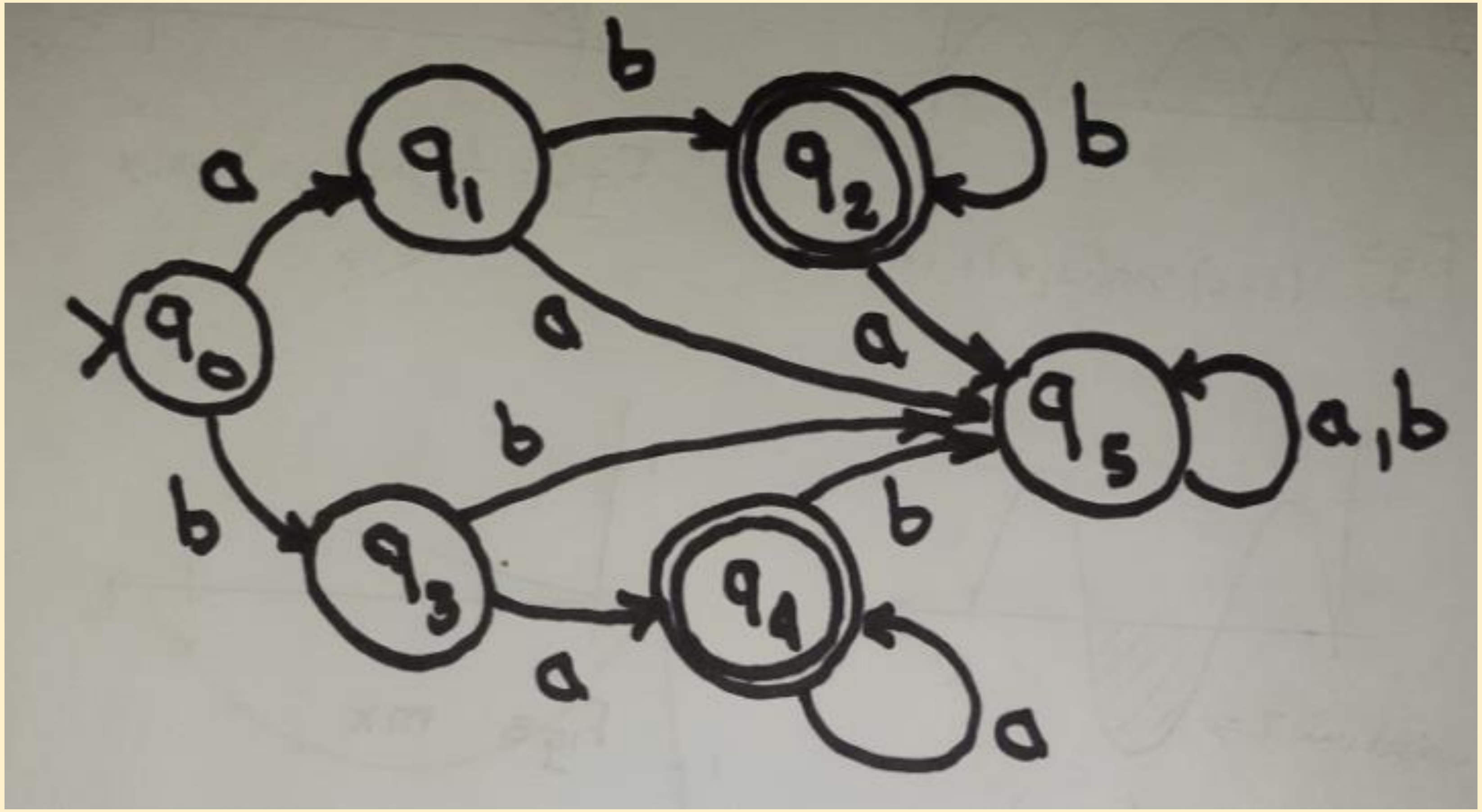
- Let $\Sigma = \{a, b\}$, and $L = (bb)^*aa(bb)^*$
- Draw the FSM for L and provide its $M = \{Q, \Sigma, q_o, F, \delta\}$ form.
  - We'll also draw an incorrect FSM and discuss.
  - Here's the incorrect one:



- Let $\Sigma = \{a, b\}$
- Let L be the set of strings that either …
  - Have exactly one ‘a’ in the string, which is at the beginning of the string, and is followed by at least one ‘b’, or
  - Have exactly one ‘b’ in the string, which is at the beginning of the string, and is followed by at least one ‘a’.

- Write L as a regular expression: $L = ab^+ + ba^+$

- Let $\Sigma = \{a, b\}$
- Let $L = ab^+ + ba^+$
- Draw the FSM, M, for L.



- Let $\Sigma = \{a, b\}$
- Let $L = ab^+ + ba^+$
- Write out $M = \{Q, \Sigma, q_o, F, \delta\}$.

$M = \{Q, q_0, \Sigma, F, \delta\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_5\}$

$\Sigma = \{a, b\}$

$q_0 = q_0$

$F = \{q_2, q_4\}$

δ:

	a	b
0	1	3
1	5	2
2	5	2
3	4	5
4	4	5
5	5	5

About that Transition Function ….

- All functions (using that term loosely) are defined by their input and output.
- Functions (using that term loosely) create a relation between the input set and the output set.
- The transition function, δ, takes as input a symbol and a state
  - This is actually an ordered pair (symbol, state).
- The transition function, δ, produces a state as output.
- Formally, $\delta(a, q_i) = q_j$
  - δ is a function as each ordered pair (a, q_i) appears at most once.
  - δ is a complete function, as each (a, q_i) appears exactly once.
  - So we can say that δ is not just a <u>relation</u>, it is a true <u>function</u>.

About that Transition Function ....

- Formally, $\delta(a, q_i) = q_j$ (defined for all symbols and states)
- Since δ is defined for symbols, how do we process strings?
- Recursion! Recursively define δ on strings as follows:
  - $\delta($ no symbol, $q_i) =$ no transition
  - $\delta(a, q_i) = q_j$ (unchanged from before)
  - $\delta(u, q_i)$ is defined as follows:
    - Let $u = au'$, where a is the first symbol in u, and u' is the rest of the string.
    - Then $\delta(u, q_i) = \delta(au', q_i) = \delta(u', \delta(a, q_i))$
    - In other words, to process a string, process the first symbol and then the rest.
- M accepts a string, u, iff $\delta(u, q_o) \in F$
  - You knew this, its just the formal notation for it.
  - This is useful in proving a given M recognizes a given L.