Computational Foundations of Informatics

Vasant G. Honavar
Edward Frymoyer Professor of Information Sciences and Technology
Artificial Intelligence Research Laboratory
Informatics Graduate Program
Computer Science and Engineering Graduate Program
Bioinformatics and Genomics Graduate Program
Neuroscience Graduate Program
Data Sciences Undergraduate Program
Center for Big Data Analytics and Discovery Informatics
Huck Institutes of the Life Sciences
Institute for Computational and Data Sciences
Clinical and Translational Sciences Institute
Northeast Big Data Hub
Pennsylvania State University

Sudha Murty Distinguished Visiting Chair of Neurocomputing and Data Science
Indian Institute of Science

What is informatics?

Informatics is the study of the structure, behavior, and interactions of natural and engineered computational systems, including genomes, cells, brains, computers, organisms, societies. Informatics is concerned with the representation, processing, and communication of information in such systems.

Introductions

- Vasant Honavar
  - Professor
    - Graduate programs in
      - Informatics
      - Computer Science and Engineering
      - Data Sciences
      - Bioinformatics and Genomics
      - Operations Research
      - Neuroscience
    - Undergraduate program in
      - Data Sciences
    - E335 Westgate Bldg
    - vhonavar@psu.edu
    - http://faculty.ist.psu.edu/vhonavar

- Thanh Le
  - Teaching Assistant
  - PhD Student, Informatics
- Students?

About the course

- Who is this course for?
  - Graduate students in informatics and related areas
- What are the prerequisites for taking the course?
  - Graduate standing in informatics or a related discipline
  - A willingness to venture out of one's comfort zone
  - Some mathematical literacy
  - Reading and listening critically
  - Writing coherently

About the course

Part 1: Computational Foundations of Informatics

- Computation and computing machines; Algorithms as recipes for processing descriptions; Church-Turing Thesis. Implications of Church-Turing Thesis. Algorithmic complexity – hard and easy problems.
- Information. Measuring the information content of descriptions. key results in information theory, entropy, divergence.
- Representation. Syntax and semantics. Example using propositional logic and probability.
- Computing machines and Grammars: Finite State Machines (regular grammars) to Turing machines
- Descriptional complexity – Kolmogorov Complexity and Applications.
- Communication: codes – their design and use.

About the course

Part 2: Algorithmic Abstractions in the Sciences and Humanities

- Church-Turing Thesis and the Emergence of Algorithmic Thinking
- Algorithmic Thinking in Cognitive Science / Artificial Intelligence: Example: Algorithmic inference
- Algorithmic Thinking in Brain Science: McCulloch Pitts Neurons and neural networks
- Algorithmic Thinking in Life Sciences: A hierarchy of models of macromolecular interaction networks
- Algorithmic Thinking in Social Sciences: Rational decision making, social network analysis
- Algorithmic Thinking in Humanities: Computational analysis and synthesis of information artifacts (literature, arts, music, history, culture).

Course objectives: What

- Upon successful completion of the course, you should be able to :
  - Demonstrate familiarity with the notions of computation, information, and communication, including their intellectual history, conceptual and theoretical foundations, and key applications
  - Develop useful algorithmic abstractions of different domains in the sciences and the humanities
  - Use the resulting abstractions to formulate and answer scientific questions e.g., How do we make rational choices? How do brains acquire, store, and process information? How do stem cells choose their fate? How do ideas spread through a population? How do cultures rise and fall? How do groups and coalitions form?
  - Leverage algorithmic thinking in multi-disciplinary, interdisciplinary, or trans-disciplinary research in informatics

Course objectives: How

- In order to achieve the learning goals, you will have to:
  - Participate in class
  - Critically read and discuss the assigned readings
  - Review, or learn as needed, the foundational ideas of computing, information, complexity, etc.
  - Work through examples of algorithmic abstractions in a variety of domains
  - Review, or learn as needed, the relevant topics in mathematics
  - As needed, familiarize yourself with the relevant tools
  - Complete the assignments

About the course

- Expectations
  - Class participation
  - Reading assignments
  - Scribing lecture notes
  - Problem sets
  - Written assignments
  - Term paper / project
  - Adherence to course polices regarding academic honesty

Course Materials

- No required textbook
- Recommended books
  - The Pattern on the Stone: The Simple Ideas that Make Computers Work, Daniel Hillis, Basic Books, 1998
  - Grammatical Man: Information, Entropy, Language, and Life, Jeremy Campbell, Simon and Schuster, 1982
  - The Sciences of the Artificial, Herbert Simon, MIT Press, 1996
  - Others to be added
- Required readings
- Recommended readings
- Lecture slides
- Lecture notes (prepared by assigned scribes, reviewed/edited by course staff)

Computing and Informatics

- Computing and informatics are often equated with the digital technologies and the ubiquitous role they play in virtually every aspect of our day-to-day lives
- Parallels are often drawn between the invention of digital technologies such as the world-wide web and Gutenberg's invention of the printing press
- History of computing and informatics is really a history of human attempts to understand ourselves and the world around us
- While the digital technologies enabled by computing are no doubt useful and perhaps even transformative, algorithmic abstractions are even more so!
  - Algorithmic abstractions
    - Increasingly assume the role played by mathematics in the sciences over a few hundred years
    - Constitute the modern lingua franca for sciences and the humanities
    - Provide an orderly, formal framework and exploratory apparatus for other sciences, and even the humanities

Computing and Informatics

- Physics allows us to look at the world using physics based abstractions
- Physics based abstractions help make sense of physical phenomena in nature
- Informatics enables us to look at the world through algorithmic abstractions
- Informatics enables us to understand how information is encoded, stored, manipulated, and used in
  - Nature – in genomes, cells, brains, organisms, organizations, societies
  - Human artifacts
    - Sciences – data, models, hypotheses, theories
    - Humanities – literature, music, drawings, paintings, history, language, culture

Computing and Informatics

- Computing and informatics increasingly play now the role played by mathematics in the sciences beginning with the early 17th century
  - Algorithmic Abstractions : Cognitive and Brain Sciences :: Calculus : Physics
- Implications
  - We will have a theory of reasoning when we have algorithms that reason
  - We will have a theory of learning when we have algorithms that learn from experience

Computing and Informatics

- Computing and informatics increasingly play now the role played by mathematics in the sciences beginning with the early 17th century
  - Algorithmic Abstractions : Life Sciences : Calculus : Physics
- Implications
  - We will have a theory of protein folding when we have an algorithm that accepts a linear sequence of amino acids as input and produces the description of the 3D structure of the protein as output
  - We will have a theory of protein-RNA interaction when we have an algorithm that predicts whether a given protein binds to a given RNA, and if so, identify the amino acid and nucleic acid residues at the interface

Computing and Informatics

- Computing and informatics increasingly play now the role played by mathematics in the sciences beginning with the early 17th century
  - Algorithmic Abstractions : Social Sciences: Calculus : Physics
- Implications
  - We will have a theory of rational decision making when we have algorithms for rational decision making based on individual or collective preferences
  - We will have a theory of social ties when we have an algorithm that predicts the formation and dissolution of social ties

An abbreviated history of computing and informatics

- Numbers and counting – (9^{th} -2^{nd} millennium BCE)
- Writing signs (3000 - 2000 BCE – Sumer, Egypt, Indus, Minoa)
- Positional number system (1900 BCE, Babylon, 500 BCE, Zapotecs)
- Alphabet and writing (1500 1200 BCE Semitic, 900 BCE Greek, 500 BCE Brahmi)
- Grammar (350 BCE – Sanskrit …)
- Tools for calculation (200 BCE – Chinese Abacus, Indian board)
- Zero, algebra and the first algorithms (400-600 CE - India, 800 CE - Arabia)
- Binary number system (400 BCE – China, 1700 CE, Europe)

An abbreviated history of computing and informatics

- Logic and inference:
  - Greece: Zeno (reductio ad absurdum, 5th century BCE), Plato (5th century BCE), Aristotle (4th century BCE)
  - India: Gautama (6th century BCE), Gotama (2nd century BCE), Nagarjuna (2nd century BCE)
  - China: Confucius (6th century BCE)
- Plato (5th century BCE)
  - What is it that can properly be called true or false?
  - What is the nature of the connection between the assumptions of a valid argument and its conclusion?
  - What is the nature of definition?

An abbreviated history of computing and informatics

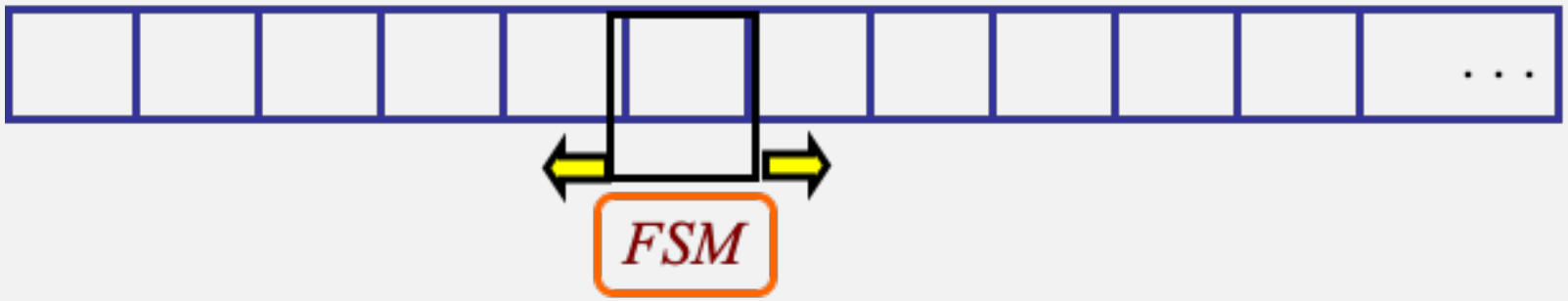
- Aristotle (4th century BCE)
  - The Categories – a study primitive terms
  - Topics – a discussion of dialectics
  - On interpretation – an analysis of simple categorical propositions
  - Prior Analytics – a formal analysis of valid arguments
  - Analytics – a study of scientific demonstration
- Al Khowarizmi (825) – diffusion of the Indian methods of numerical and algebraic calculation to the west through his text which was translated to Latin (in 12th century) as Algoritmi de numero Indorum
- Descartes (1556-1650) – Cogito ergo sum!
- Hobbs (1650) – thinking is a rule-based process analogous to arithmetic
- Leibnitz (1646-1716) seeks a general method (logical calculus) for reducing all truths to a kind of calculation

An abbreviated history of computing and informatics

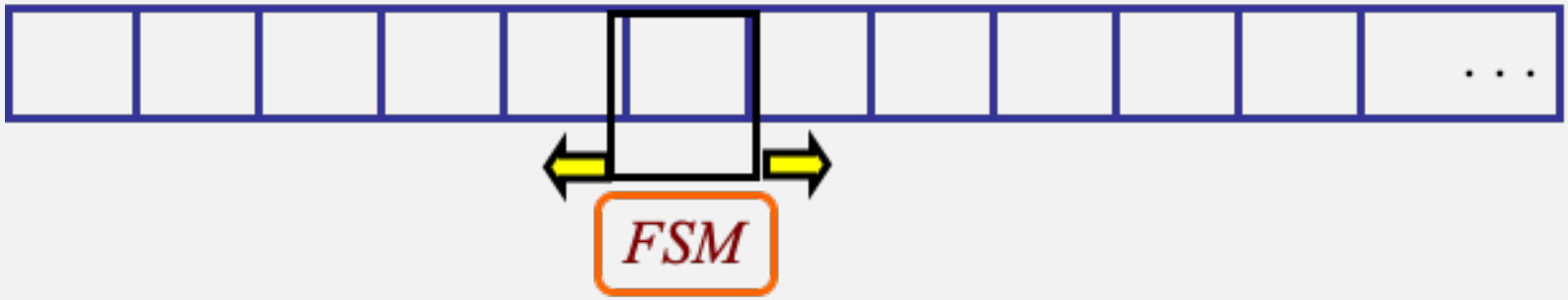
- Boole (1815-1864) develops algebraic approaches to logic and proposes logic and probability as the basis of laws of thought
- Peirce (1839-1914) introduces theory of signs, discovers many key results in logic, and lays foundations of axiomatic set theory
- Frege (1848-1925) invents axiomatic predicate logic (Begriffsschrift – concept script)
- Russell and Whitehead (1910-1915) reduce large portions of mathematics to logic
- Hilbert (1862-1943), Lowenheim (1878-1957), Skolem (1887-1963) develop mathematical logic
- Gödel (1906-1978) proves the incompleteness theorem

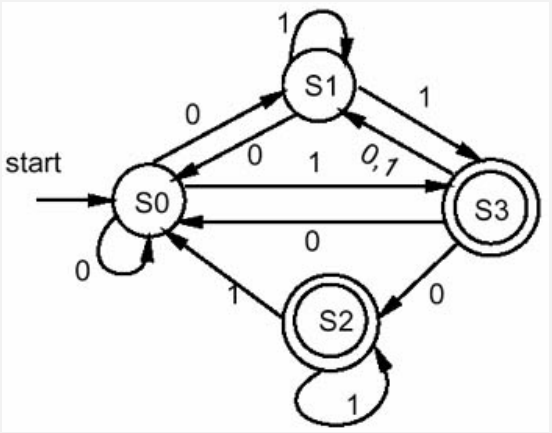
An abbreviated history of computing and informatics

- Hilbert (1862-1943) presents the decision problem – Is there an effective procedure for determining whether or not a given theorem logically follows from a given set of axioms?
- In an attempt answer Hilbert's decision problem, Turing (1912-1954) invents the Turing Machine to formalize the notion of an effective procedure
  - ✓An effective procedure is a recipe for transforming a string of letters into another string of letters
  - ✓A recipe (program) executed by the Turing machine



Turing Machine





7-tuple: $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$

Q: finite set of states

Σ: input alphabet (cannot include blank symbol, _)

Γ: tape alphabet, includes Σ and _

δ: transition function: $Q \times \Gamma \rightarrow Q \times \Gamma \times \{\mathbf{L}, \mathbf{R}\}$

q_0: start state, $q_0 \in Q$

q_{accept}: accepting state, $q_{accept} \in Q$

q_{reject}: rejecting state, $q_{reject} \in Q$

Turing machine offers a universal description system

- Turing Machine formalizes the notion of an effective procedure
  - ✓A program executed by the Turing machine that transforms one string of letters into another
- Answer to Hilbert's decision problem
  - Qualified yes - use a Turing machine program
    - The program decides that the conclusion follows from assumptions if indeed it does
    - Otherwise, the program may not terminate
- Church, Kleene, Post, Markov (1930-1950) develop other models of computation based on alternative formalizations of effective procedures which proved to be Turing-equivalent

Universal description systems

- Turing and Church put forth the Church-Turing thesis
  - Anything that is computable is computable by a Turing Machine!
  - Anything that is describable is describable by a computer program!
- Turing machines are universal computers!
- Anything that is computable is computable by a Turing Machine!
- Anything that is describable is describable by a computer program!
- Computation is the best formalism we have for describing how information is processed in
  - Computers
  - Brains
  - Genomes
  - Organizations
  - societies …

Emergence of algorithmic abstractions

- Informatics is a science of information processing
- Church-Turing Thesis
  - Anything that is describable can be described by a computer program
- Computing offers
  - A universal medium for creation, representation, analysis, synthesis, communication, of information-rich artifacts
    - Data and knowledge, scientific theories
    - History, culture, arts, music, and literature
  - An orderly formalism and exploratory apparatus for describing the processes that synthesize, organize, transform, extract, abstract … information … from information-rich artifacts

Example of Algorithmic Abstractions: Artificial Intelligence

Computation: Cognition: : Calculus : Physics

- How do we perceive?
- How do we learn?
- How do we reason?
- How do we make decisions?
- How do we plan and act?
- How do we create?
- How do we communicate?
- We will have a theory of learning when we have
  - ✓ precise information processing models of learning
  - ✓ computer programs that learn from experience

Example of Algorithmic Thinking: Life Sciences

Computation: Life : : Calculus : Physics

- How do cells respond to their local environment?
- How do brains acquire, process, and store information?
- How do genes and environment determine behavior?
- How do stem cells choose their fate?
- How do organs develop?
- What determines the 3-dimensional structure of a protein?
- What differentiates a healthy cell from a cancerous cell?
- How do macromolecules recognize each other?
- We will have a theory of protein folding when we have an algorithm that accepts a linear sequence of amino acids as input and produces a description of the 3-dimensional structure of a protein as output

Example of Algorithmic Thinking: Social sciences

Computation: Society : : Calculus : Physics

- What is rational behavior?
- Under what conditions do self-interested rational agents cooperate to achieve a common good?
- How do groups and coalitions form?
- How do different organizations process information?
- How to balance thinking versus acting?
- How can agents coordinate their behavior?
- How do organizations evolve?
- How do ideas spread through a population?
- How do cultures rise and fall?
- We will have a theory of ….

Problems and machines that solve them

Main Question

What *problems* can a particular type of *machine* solve?

What is a *problem*?

What is a *machine*?

What does it mean for a *machine to solve a problem*?

What exactly do we mean by a problem?

A **problem** is defined by:

- A description of a set of **inputs**
- A description of a set of outputs and the property an **output** must have

A machine **solves** a problem if for *every* **input** it *eventually* produces a satisfactory **output**.

Outputs

How many possible outputs do you need for a problem to be interesting?

2 – "Yes" or "No"

A ***decision* problem** is a problem that has two possible outputs.

Finite search problems can be framed as a series of decision problems:

What is **2+2**? vs. Does **2+2=0**? **No**

Does **2+2=1**? **No**

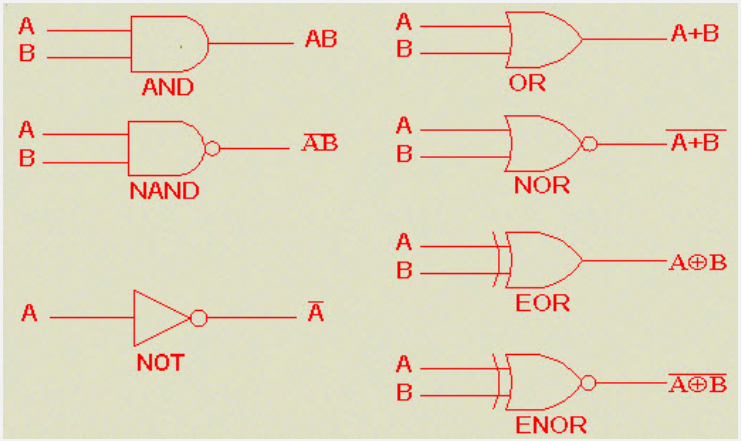
Does **2+2=2**? **No**

Does **2+2=3**? **No**

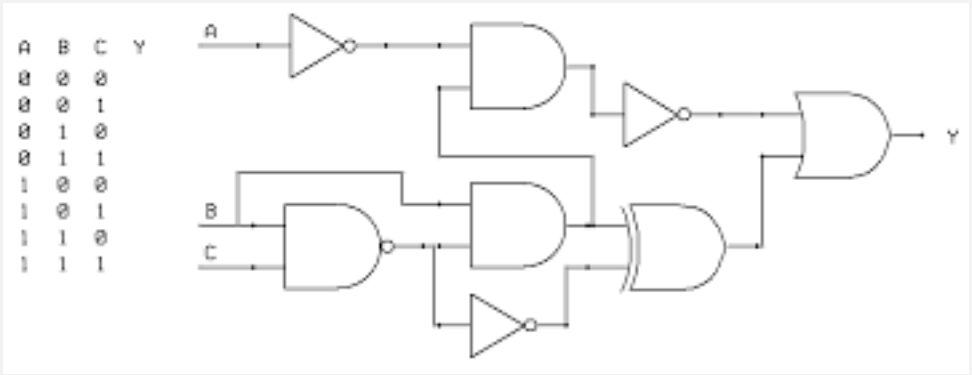
Does **2+2=4**? **Yes**

Simple problems

- Finite inputs, finite outputs
- Example, functions computed by logic circuits



Fact: You can compute any Boolean Logic Function using a combination of AND, OR, and NOT



Language Recognition

A language is a (finite or infinite) set of strings over some finite alphabet.

Is string s in language L?

A machine M **recognizes** language L if it can answer: for any string s, is s in L?

We can describe the **power** of a type of machine is by the **set of languages it can recognize**.

Languages, Grammars, and Computers

- Grammars have an intimate relationship with computing devices
- Each class of languages in the Chomsky hierarchy (1956)[1] corresponds to a class of computing devices
  - Finite state machines
  - Push-down automata
  - Linear bounded non-deterministic Turing machine
  - Turing machines
- Natural languages can arguably be dealt with using finite state machines[2]

[1]Chomsky, Noam (1956). "Three models for the description of language". IRE Transactions on Information Theory (2): 113–124.

[2]Borgida, A. T. (1983.) Some formal results about stratificational grammars and their relevance to linguistics. Mathematical Systems Theory 16, 29-56

Grammars

- A formal grammar of this type consists of a finite set of production rules (left-hand side → right-hand side), where each side consists of a finite sequence of:
  - a finite set of nonterminal symbols e.g., *A, B, S*
  - a finite sequence of terminal ot non-terminal symbols

  e.g., $\propto, \beta, \gamma, \delta$ (where γ cannot be empty)
  - a start symbol (a distinguished nonterminal symbol) *S*

Example:

A grammar specified by terminals {*a, b*}, Nonterminals {*S*}, Start symbol *S*, Production rules

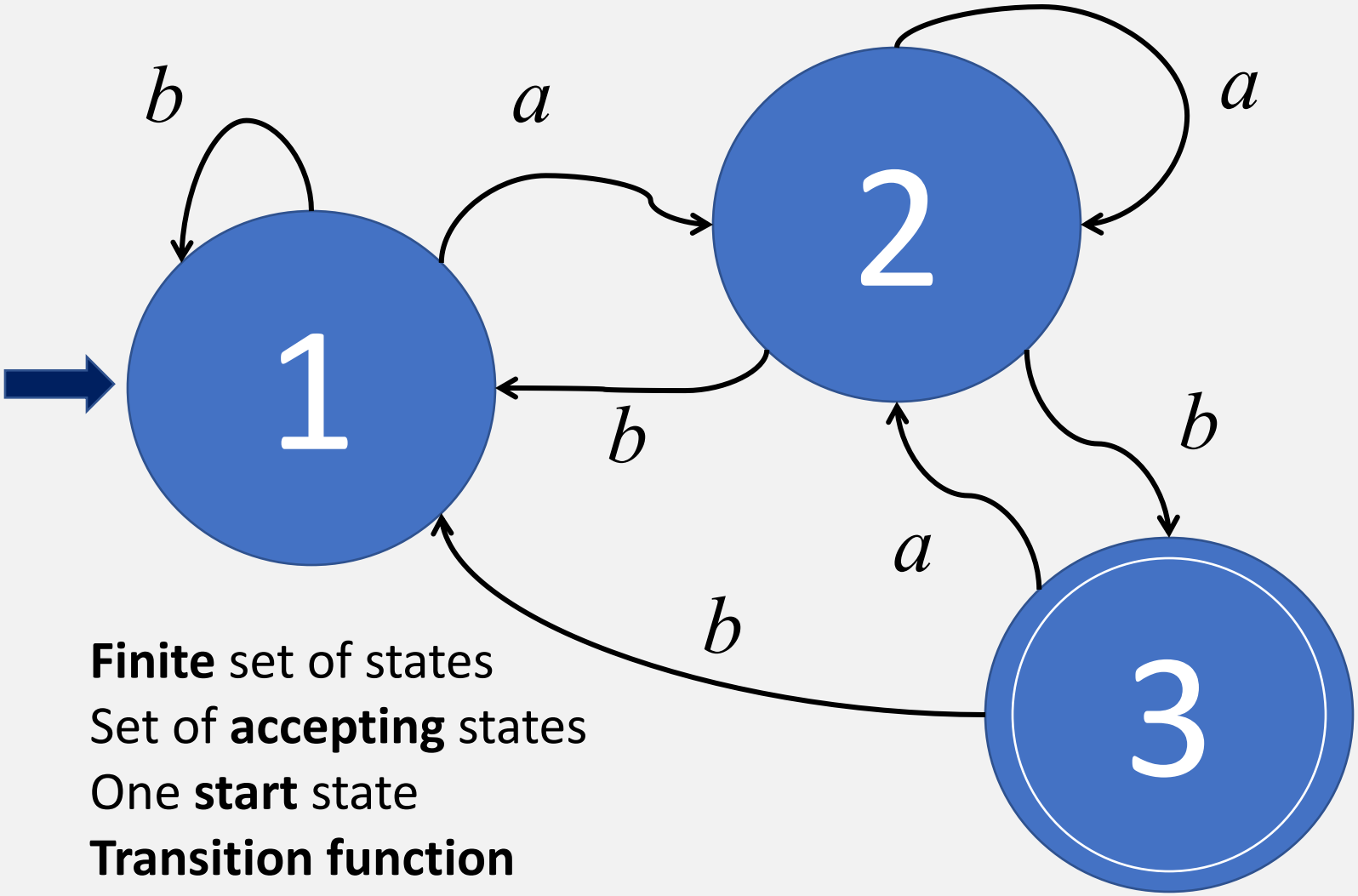
- $S \rightarrow aSb$
- $S \rightarrow \varepsilon$

Specifies the language $\{a^n b^n | n \geq 0\}$

Chomsky Hierarchy

- Regular languages (Type 3)
  - Rules of the form: $A \rightarrow aB$; $A \rightarrow a$
  - Example language: $\{a^n | n \geq 0\}$
  - Recognized by: Finite State Machines
- Context-free languages (Type 2)
  - Rules of the form: $A \rightarrow \alpha$
  - Example language: $\{a^n b^n | n > 0\}$
  - Recognized by: Pushdown automata
- Context-sensitive languages (Type 1)
  - Rules of the form: $\alpha A \beta \rightarrow \alpha \gamma \beta$
  - Example language:$\{a^n b^n c^n | n > 0\}$
  - Recognized by: Linear bounded Turing Machines
- Unrestricted languages (Type 0)
  - Recognized by Turing machines

Deterministic Finite Automata



Finite set of states
Set of **accepting** states
One **start** state
Transition function

Deterministic Finite Automata

Definition: A deterministic finite automaton (DFA) consists of

1. a finite set of *states* (often denoted Q)
2. a finite set Σ of *symbols* (alphabet)
3. a *transition function* that takes as argument a state and a symbol and returns a state (often denoted δ)
4. a *start state* often denoted q_0
5. a set of *final* or *accepting* states (often denoted F) We have $q_0 \in Q$ and $F \subseteq Q$

Deterministic Finite Automata

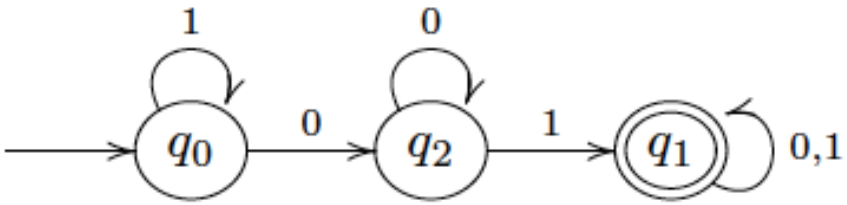
How to present a DFA? With a *transition table*

	0	1
$\rightarrow q_0$	q_2	q_0
$*q_1$	q_1	q_1
q_2	q_2	q_1

The $\rightarrow$ indicates the *start* state: here q_0

The $*$ indicates the final state(s) (here only one final state q_1)

This defines the following *transition diagram*



Deterministic Finite Automata

For this example

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

start state q_0

$F = \{q_1\}$

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

δ is a *function* from $Q \times \Sigma$ to Q

$\delta : Q \times \Sigma \rightarrow Q$

$\delta(q_0, 1) = q_0$

$\delta(q_0, 0) = q_2$

How a DFA Processes Strings

Let us build an automaton that accepts the words that contain 01 as a subword

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

$L = \{x01y \mid x, y \in \Sigma^*\}$

We use the following states

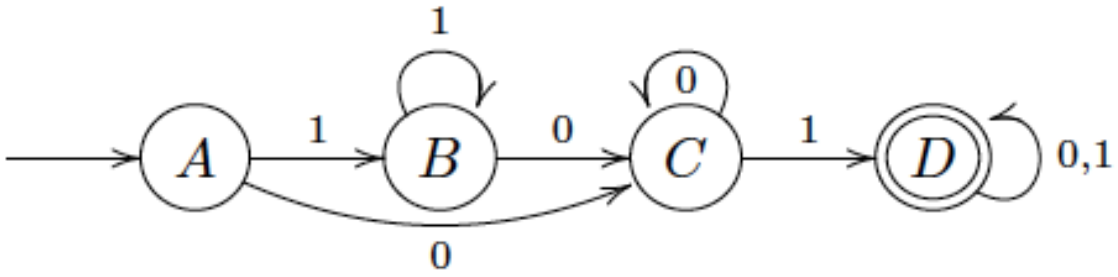
A: start

B: the most recent input was 1 (but not 01 yet)

C: the most recent input was 0 (so if we get a 1 next we should go to the accepting state D)

D: we have encountered 01 (accepting state)

We get the following automaton

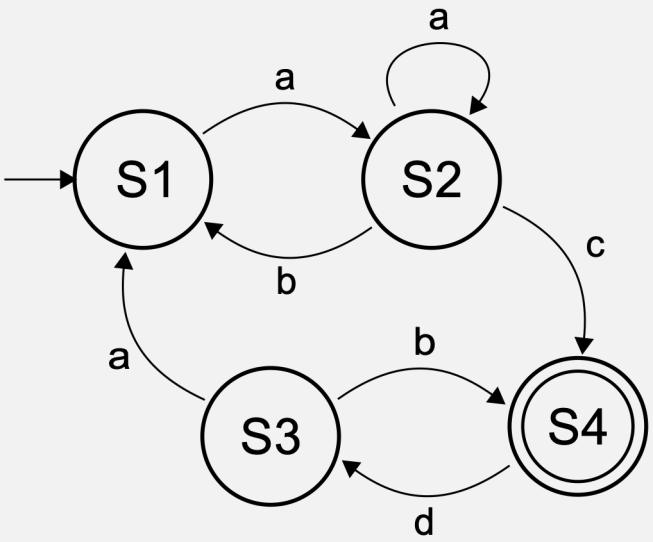


Transition table

	0	1
→A	C	B
B	C	B
C	C	D
*D	D	D

$Q = \{A,B,C,D\}$, $\Sigma = \{0,1\}$, start state A, final state(s) {D}

Example



Which of the strings below belong to the language accepted by the FSM shown?

- aaacdb
- ababacdaaac
- abcdb
- acda
- acdbdb

Extending the Transition Function to Strings

In the previous example, what happens if we get 011? 100? 10101?

We define $\hat{\delta}(q, x)$ by induction

$\hat{\delta} : Q \times \Sigma^* \rightarrow Q$

BASIS $\hat{\delta}(q, \epsilon) = q$ for $|x| = 0$

INDUCTION suppose $x = ay$ (y is a string, a is a symbol)

$\hat{\delta}(q, ay) = \hat{\delta}(\delta(q, a), y)$

Notice that if $x = a$ we have

$\hat{\delta}(q, a) = \delta(q, a)$ since $a = a\epsilon$ and $\hat{\delta}(\delta(q, a), \epsilon) = \delta(q, a)$

Extending the Transition Function to Strings

$\hat{\delta} : Q \times \Sigma^* \rightarrow Q$

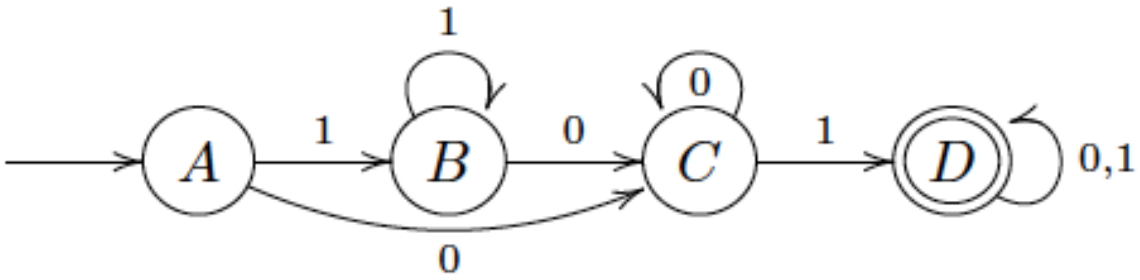
We write $q.x$ instead of $\hat{\delta}(q, x)$

We can now define mathematically the *language* accepted by a given automaton $Q, \Sigma, \delta, q_0, F$

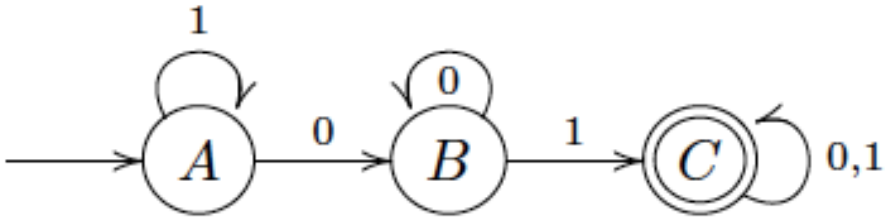
$L = \{x \in \Sigma^* \mid q_0.x \in F\}$

On the previous example 100 is not accepted and 10101 is accepted

The same language may be represented by different DFA



and



- There is a minimal DFA
- There is an algorithm for minimizing a DFA

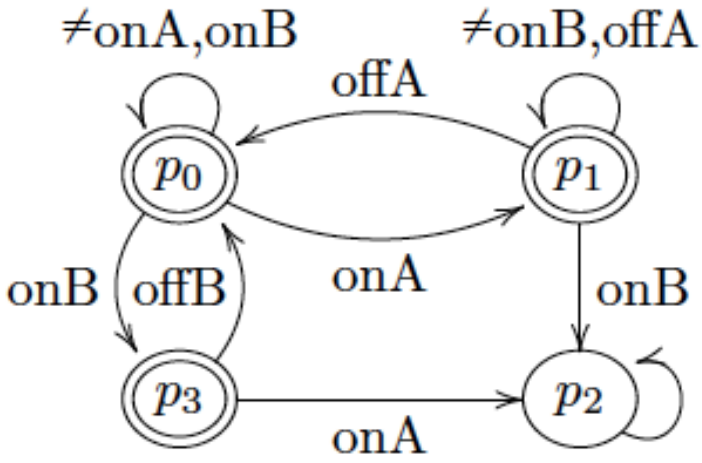
Application: control system

We have several machines working concurrently

We need to forbid some sequence of actions. For instance, if we have two machines MA and MB, we may want to say that MB cannot be on when MA is on. The alphabets will contain: onA, offA, onB, offB

Between onA, on2 there should be at least one offA

The automaton expressing this condition is



Question

What languages can be recognized by a DFA?

The ***regular languages***.

A language is a regular language if there is some DFA that recognizes it.

Questions?

Can all finite languages be recognized by a DFA?

Yes. Trivially: create a state-path for each string in the language. Finite language, means a finite number of states is enough.

Can a DFA recognize an infinite language?

Yes. We've seen an example already!

Are there languages a DFA cannot recognize?

Yes. For example, the sentences that are palindromes!

Formal Definition

A finite automaton is a 5-tuple:

Q **finite** set ("states")

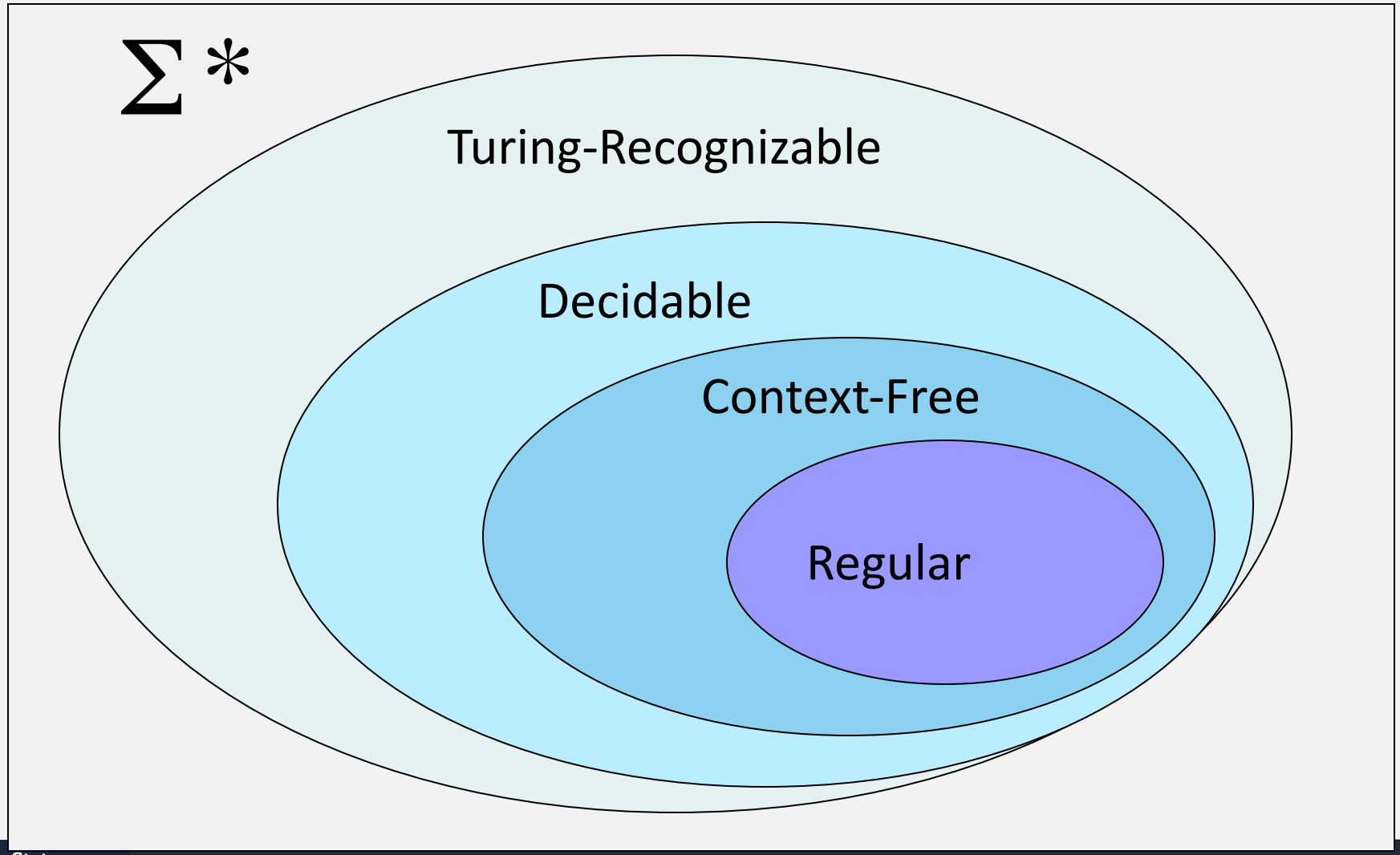
Σ **finite** set ("alphabet")

$\delta : Q \times \Sigma \rightarrow Q$ **transition function**

$q_0 \in Q$ start state

$F \subseteq Q$ set of accepting states

Chomsky Hierarchy

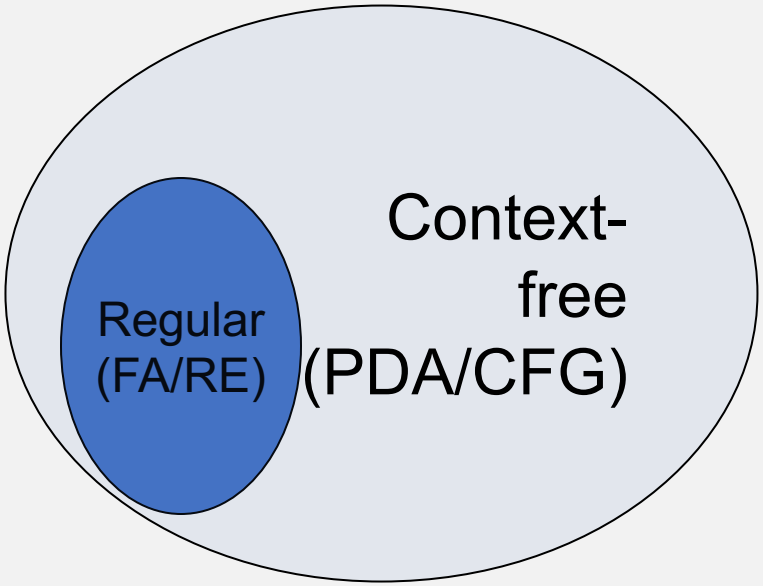


Not all languages are regular

- So what happens to the languages which are not regular?
- Can we still come up with a language recognizer?
  - i.e., something that will accept (or reject) strings that belong (or do not belong) to the language?

Context-Free Languages

- A language class larger than the class of regular languages
- Supports natural, recursive notation called “context-free grammar”
- Applications:
  - Parse trees
  - ..



An Example

- A palindrome is a word that reads identical from both ends
  - E.g., madam, redivider, malayalam, 010010010
- Let L = { w | w is a binary palindrome}
- Is L regular?
  - No.
  - Proof: Omitted

But the language of palindromes…

is a CFL, because it supports recursive substitution (in the form of a CFG)

- This is because we can construct a *"grammar"* like this:

1. A ==> ε
2. A ==> 0
3. A ==> 1
4. A ==> 0A0
5. A ==> 1A1

Terminal

Productions

Variable or non-terminal

Same as:
A => 0A0 | 1A1 | 0 | 1 | ε

How does this grammar work?

How does the CFG for palindromes work?

An input string belongs to the language (i.e., accepted) iff it can be generated by the CFG

- Example: w=01110
- G can generate w as follows:

1. A => 0A0
2. => 01A10
3. => 01110

G:
A => 0A0 | 1A1 | 0 | 1 | ε

Generating a string from a grammar:

1. Pick and choose a sequence of rewrite rules that would allow us to generate the string.
2. At every step, substitute one variable with one of its productions.

Context-Free Grammar: Definition

- A context-free grammar G=(V,T,P,S), where:
  - V: set of variables or non-terminals
  - T: set of terminals (= alphabet U {ε})
  - P: set of *productions,* each of which is of the form
    V --> α_1 | α_2 | …
    - Where each α_i is an arbitrary string of variables and terminals
  - S is the start variable

CFG for the language of binary palindromes:

G=({A},{0,1},P,A)

P: A --> 0 A 0 | 1 A 1 | 0 | 1 | ε

More examples

- Syntax checking
- In scenarios where there is a general need for:
  - Matching a symbol with another symbol, or
  - Matching a count of one symbol with that of another symbol, or
  - Recursively substituting one symbol with a string of other symbols

Context-Free Language

- The language of a CFG, G=(V,T,P,S), denoted by L(G), is the set of terminal strings that have a derivation from the start variable S.
  - $L(G) = \{ w \text{ in } T^* \mid S \Rightarrow^*_G w \}$

Pushdown Automata recognize context-free languages

- A PDA P := (Q,∑,Γ, δ,q_0,Z_0,F):
  - Q: states of the ε-NFA
  - ∑: input alphabet
  - Γ : stack symbols
  - δ: transition function
  - q_0: start state
  - Z_0: Initial stack top symbol
  - F:Final/accepting states

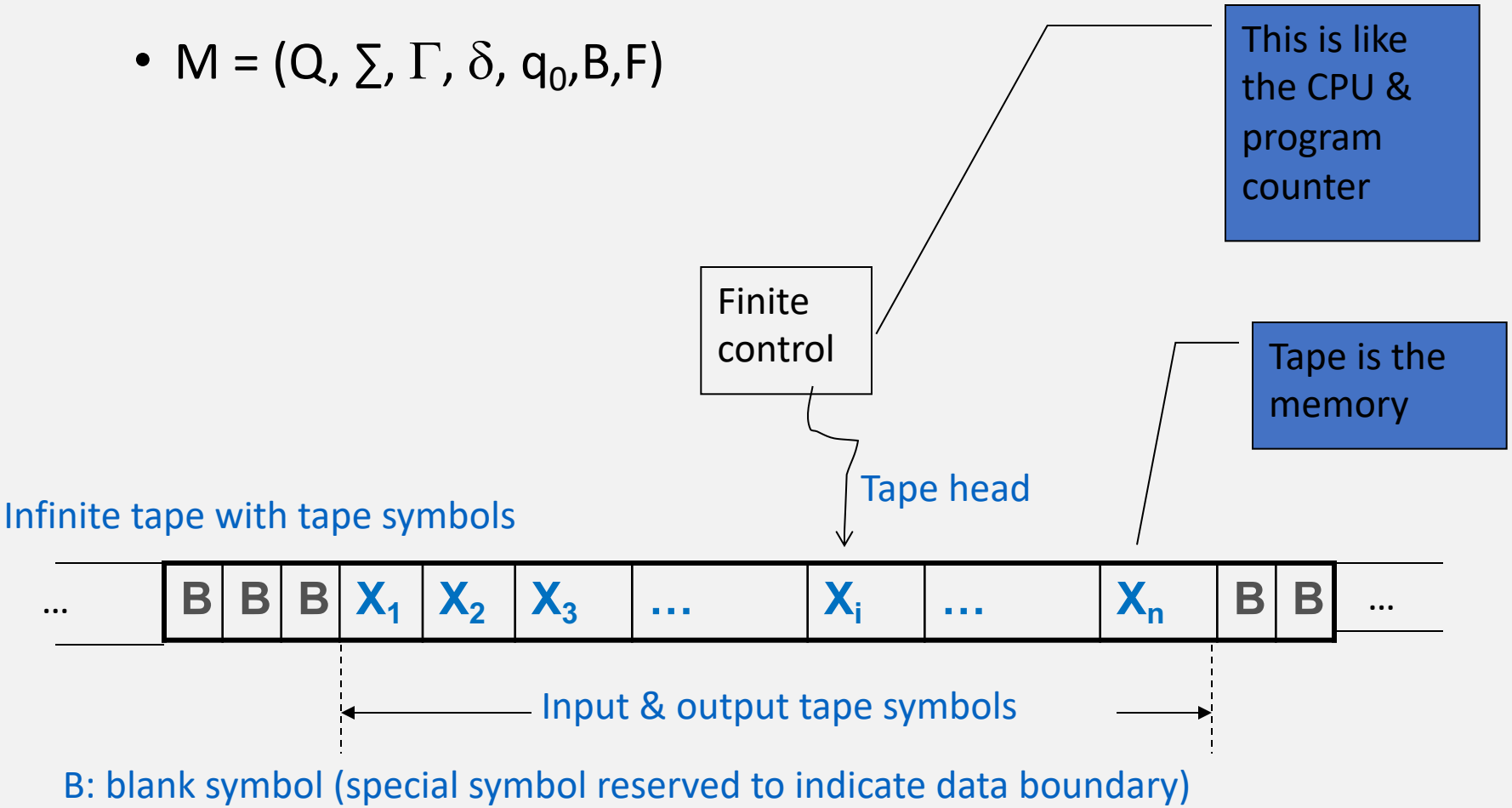
Turing Machines are…

- Very powerful (abstract) machines that could simulate any modern day computer (although very, very slowly!)

- Why design such a machine?
  - If a problem cannot be "solved" even using a TM, then it implies that the problem is ***undecidable***

A Turing Machine (TM)

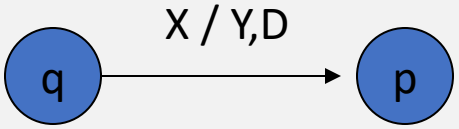
- $M = (Q, \sum, \Gamma, \delta, q_0, B, F)$



You can also use:
➔ for R
🡐 for L

Transition function

- One move (denoted by |---) in a TM does the following:
  - $\delta(q,X) = (p,Y,D)$
    - q is the current state
    - X is the current tape symbol pointed by tape head
    - State changes from q to p
    - <u>After the move:</u>
      - X is replaced with symbol Y
      - If D="L", the tape head moves "left" by one position.
        Alternatively, if D="R" the tape head moves "right" by one position.



ID of a TM

- Instantaneous Description or ID :
  - $X_1X_2\ldots X_{i-1}qX_iX_{i+1}\ldots X_n$ means:
    - q is the current state
    - Tape head is pointing to X_i
    - $X_1X_2\ldots X_{i-1}X_iX_{i+1}\ldots X_n$ are the current tape symbols

- $\delta(q,X_i) = (p,Y,R)$ is same as:

  $X_1\ldots X_{i-1}qX_i\ldots X_n \;|\text{----}\; X_1\ldots X_{i-1}YpX_{i+1}\ldots X_n$

- $\delta(q,X_i) = (p,Y,L)$ is same as:

  $X_1\ldots X_{i-1}qX_i\ldots X_n \;|\text{----}\; X_1\ldots pX_{i-1}YX_{i+1}\ldots X_n$

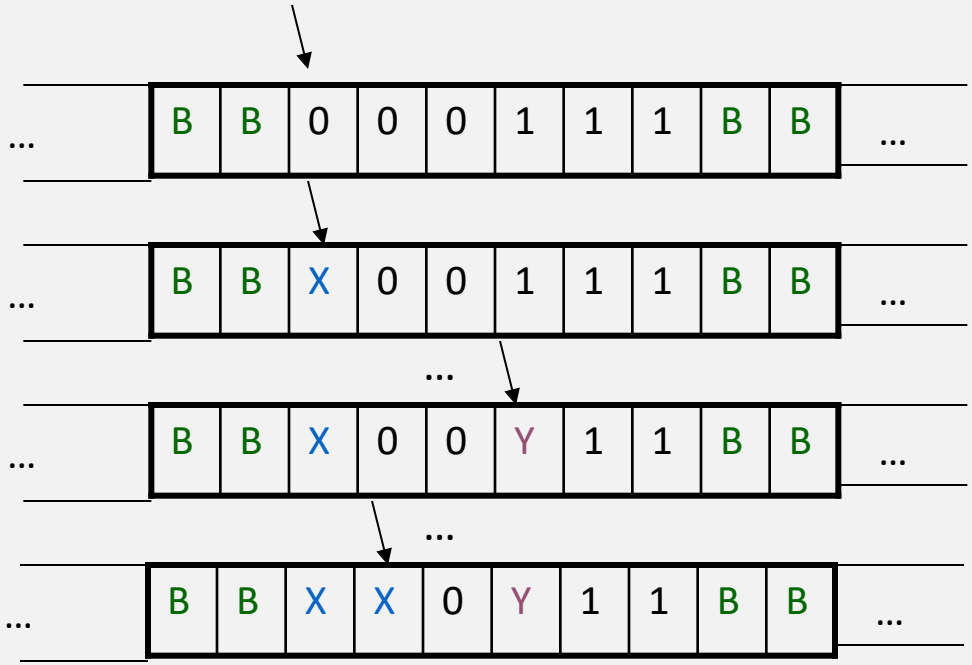
Way to check for Membership

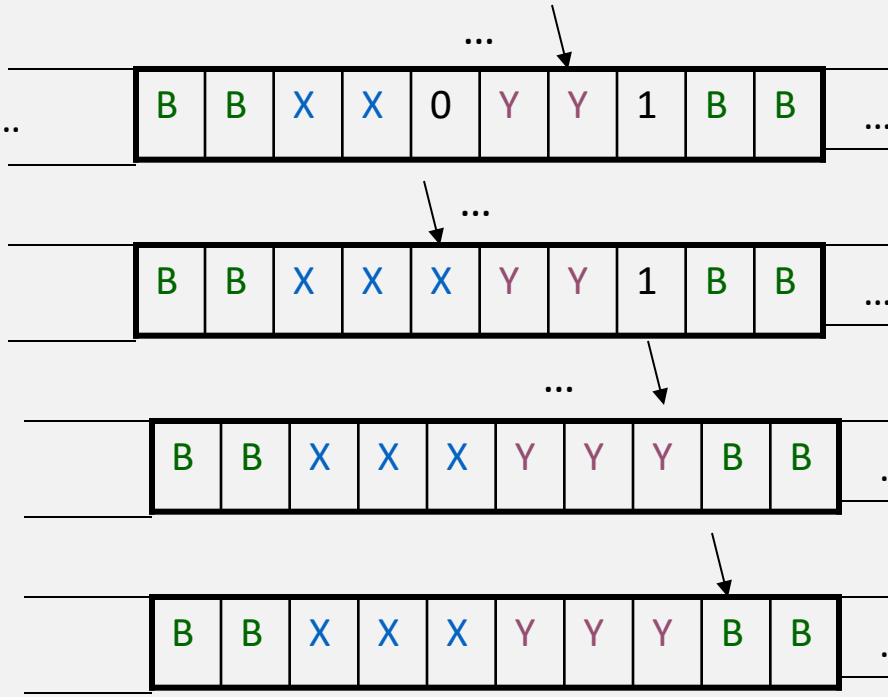
- Is a string *w* accepted by a TM?

- Initial condition:
  - The (whole) input string *w* is present in TM, preceded and followed by infinite blank symbols
- Final acceptance:
  - Accept *w* if TM enters final state and halts
  - If TM halts and not final state, then reject

Example: L = {0^n1^n | n≥1}

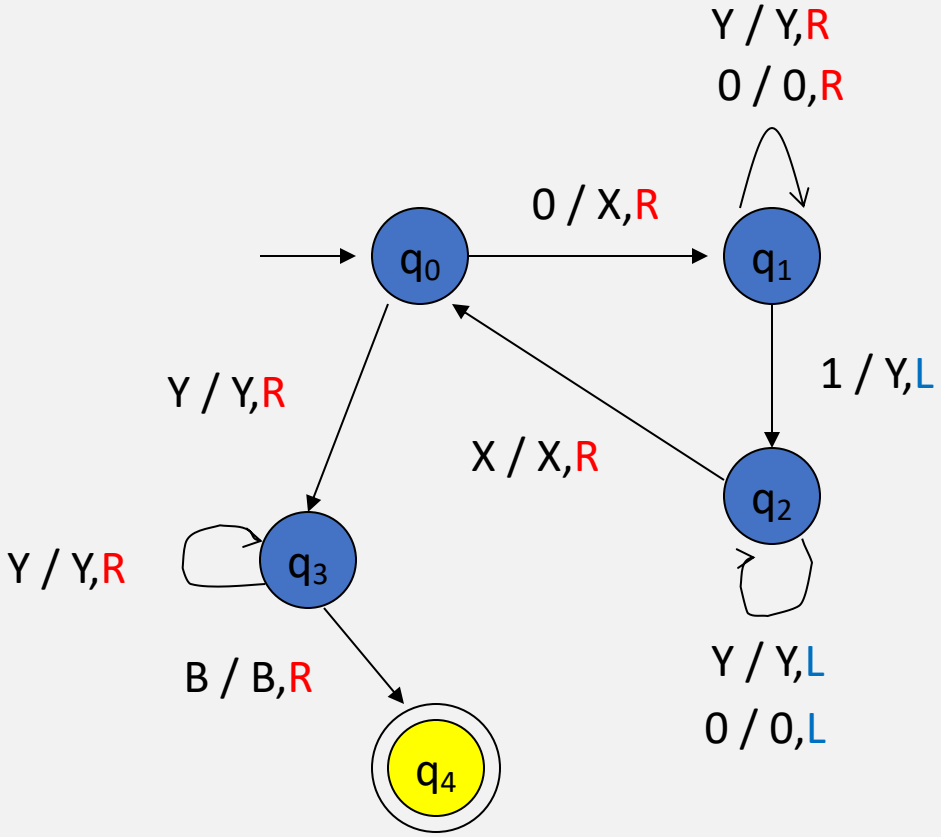
- Strategy: w = 000111





Accept

TM for $\{0^n1^n \mid n \geq 1\}$



1. Mark next unread 0 with X and move right
2. Move to the right all the way to the first unread 1, and mark it with Y
3. Move back (to the left) all the way to the last marked X, and then move one position to the right
4. If the next position is 0, then goto step 1.
   Else move all the way to the right to ensure there are no excess 1s.
   If not move right to the next blank symbol and stop & accept.

* state diagram representation preferred

TM for $\{0^n1^n \mid n \geq 1\}$

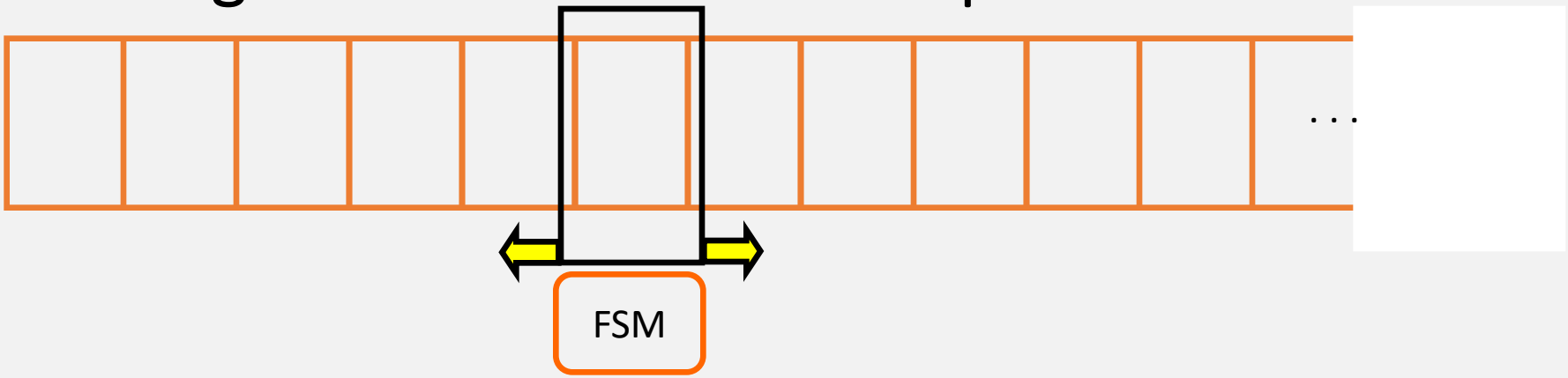
	Next Tape Symbol				
Curr. State	0	1	X	Y	B
→ q_0	(q_1,X,R)	-	-	(q_3,Y,R)	-
q_1	$(q_1,0,R)$	(q_2,Y,L)	-	(q_1,Y,R)	-
q_2	$(q_2,0,L)$	-	(q_0,X,R)	(q_2,Y,L)	-
q_3	-	-	-	(q_3,Y,R)	(q_4,B,R)
$^*q_4$	-	--	-	-	-

Table representation of the state diagram

TMs for calculations

- TMs can also be used for calculating values
  - Like arithmetic computations
  - Eg., addition, subtraction, multiplication, etc.

Turing Machine Formal Description



7-tuple: $(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$

Q: finite set of states

Σ: input alphabet (cannot include blank symbol, _)

Γ: tape alphabet, includes Σ and _

δ: transition function: $Q \times \Gamma \rightarrow Q \times \Gamma \times \{\mathbf{L}, \mathbf{R}\}$

q_0: start state, $q_0 \in Q$

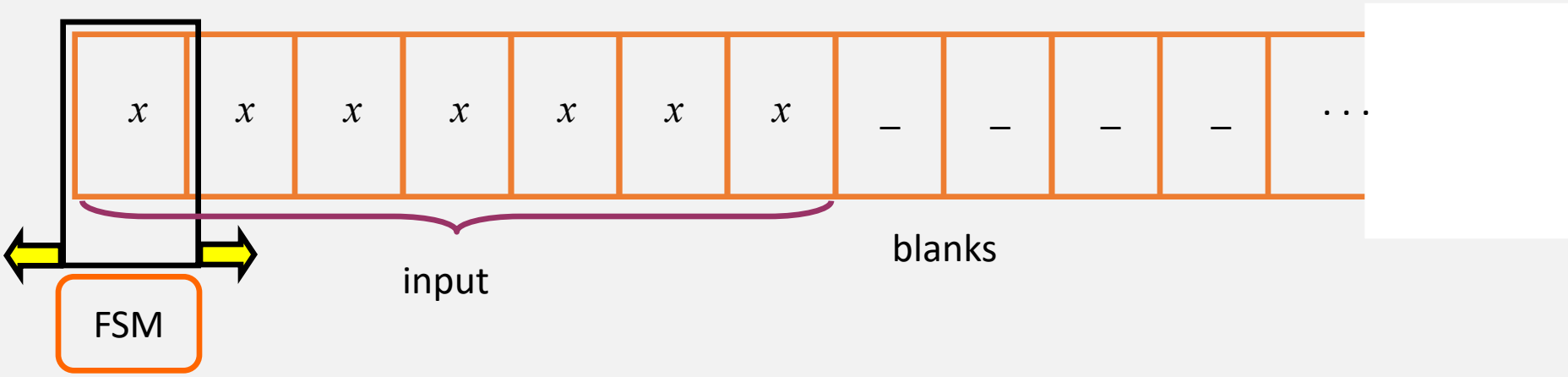
q_{accept}: accepting state, $q_{\text{accept}} \in Q$

q_{reject}: rejecting state, $q_{\text{reject}} \in Q$

Turing Machine Computing Model

Initial configuration:

$x \in \Sigma$



TM Configuration: $\Gamma^* \times Q \times \Gamma^*$

tape contents left of head

current FSM state

tape contents head and right

TM Computing Model

$$\delta^*: \Gamma^* \times Q \times \Gamma^* \rightarrow \Gamma^* \times Q \times \Gamma^*$$

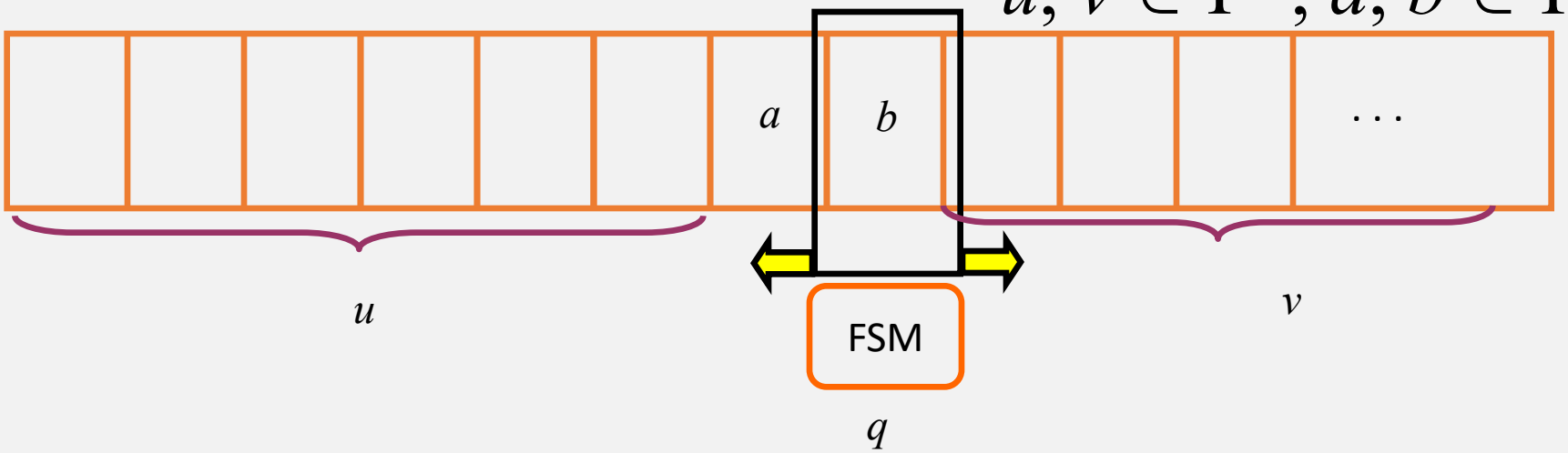
The q_{accept} and q_{reject} states are final:

$$\delta^*(L, q_{\text{accept}}, R) \rightarrow (L, q_{\text{accept}}, R)$$
$$\delta^*(L, q_{\text{reject}}, R) \rightarrow (L, q_{\text{reject}}, R)$$

TM Computing Model

$$\delta^*: \Gamma^* \times Q \times \Gamma^* \rightarrow \Gamma^* \times Q \times \Gamma^*$$

$$u, v \in \Gamma^*, a, b \in \Gamma$$



$$\delta^*(ua, q, bv) = (uac, q_r, v) \text{ if } \delta(q, b) = (q_r, c, \mathbf{R})$$

$$\delta^*(ua, q, bv) = (u, q_r, acv) \text{ if } \delta(q, b) = (q_r, c, \mathbf{L})$$

Also: need a rule to cover what happens at left edge of tape

Calculations vs. Languages

A "calculation" is one that takes an input and outputs a value (or values)

The "language" for a certain calculation is the set of strings of the form "<input, output>", where the output corresponds to a valid calculated value for the input

A "language" is a set of strings that meet certain criteria

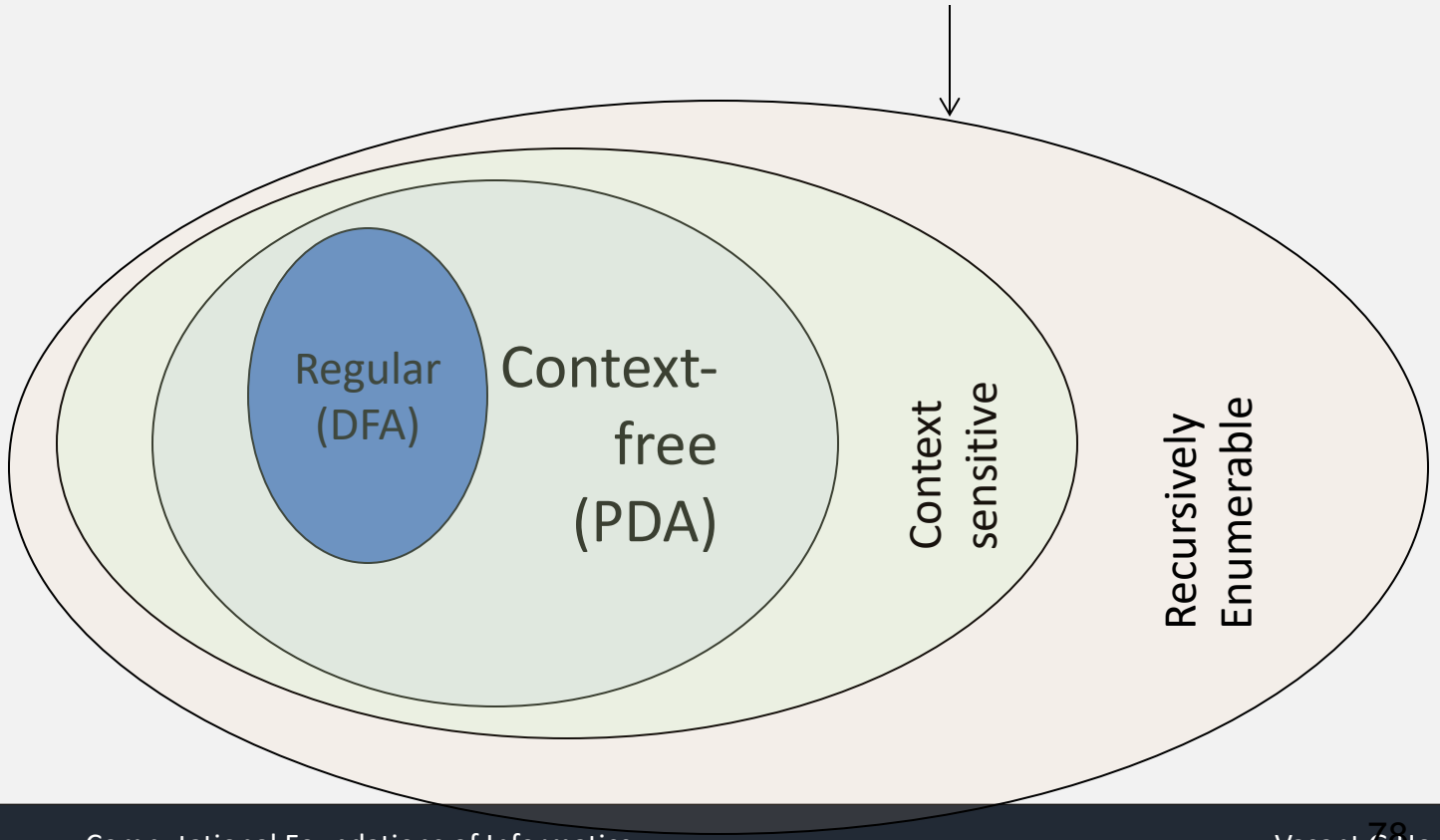
E.g., The language L_{add} for the addition operation

"<0#0,0>"
"<0#1,1>"
…
"<2#4,6>"
…

Membership question == verifying a solution
e.g., is "<15#12,27>" a member of L_{add} ?

Language of the Turing Machines

- *Recursive Enumerable (RE) language*



Languages, Grammars, and Computers

- Grammars have an intimate relationship with computing devices
- Each class of languages in the Chomsky hierarchy (1956)[1] corresponds to a class of computing devices
  - Finite state machines
  - Push-down automata
  - Linear bounded non-deterministic Turing machine
  - Turing machines
- Natural languages can arguably be dealt with using finite state machines[2]

[1]Chomsky, Noam (1956). "Three models for the description of language". IRE Transactions on Information Theory (2): 113–124.

[2]Borgida, A. T. (1983.) Some formal results about stratificational grammars and their relevance to linguistics. Mathematical Systems Theory 16, 29-56

Algorithm

- A recipe for solving a problem
- A recipe for transforming a description into another with the desired properties
- Some problems are solvable, others not
- Some problems are easy, others hard
  - as measured by the time, and/or space needed