Informatics 1

Computation and Logic

Lecture 17

The Big Ideas

Creative Commons License

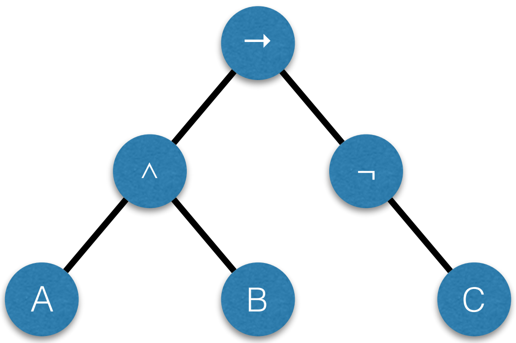
Informatics 1: Computation and Logic by Michael Paul Fourman is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

A formal language, without the vagueness and ambiguity of natural language

Syntax:
expressions are built up from atomic propositions using logical connectives

$\wedge \vee \neg \rightarrow$

expressions are trees, with atomic propositions as leaf nodes and other nodes labelled with connectives

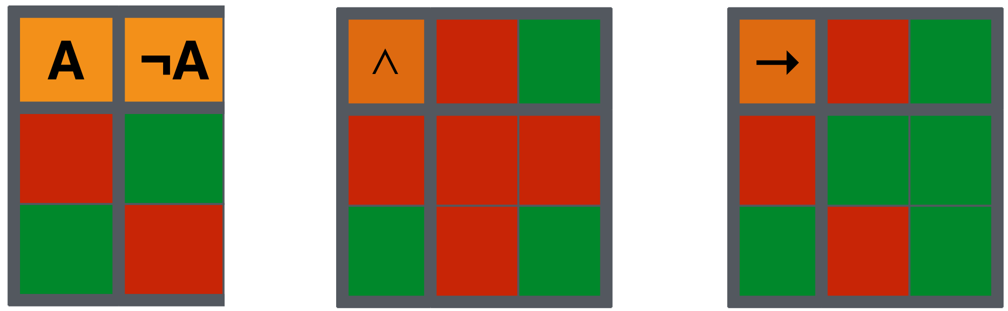


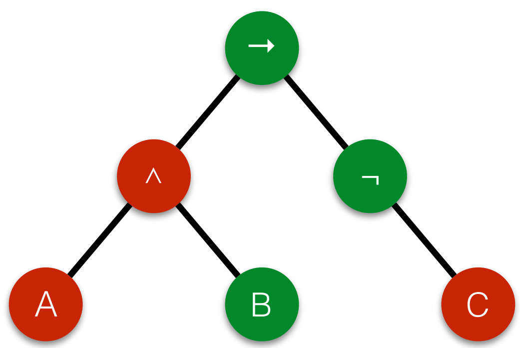
$A \wedge B \rightarrow \neg C$

precedence
$\neg$
$\wedge$
$\vee$
$\rightarrow$
$\leftrightarrow$

A formal language, without the vagueness and ambiguity of natural language

Semantics:
the truth value expressions is built up "compositionally" from the truth values of its atomic propositions using logical operators





$A \wedge B \rightarrow \neg C$

$A \wedge B \rightarrow \neg C$

$A \wedge B \rightarrow \neg C$

Formal Inference

proofs are built from assumptions using sound rules
proofs are trees, with assumptions as leaves, and other nodes labelled with instances of rules

a deduction rule

$$\frac{A \rightarrow B \quad \neg B}{\neg A}$$

an entailment

$A \rightarrow B, \neg B \vdash \neg A$

Formal Inference

proofs are built from assumptions using sound rules

proofs are trees, with assumptions as leaves, and other nodes labelled with instances of rules

a proof

$$\dfrac{A \to B \qquad \dfrac{B \to C \qquad \neg C}{\neg B}}{\neg A}$$

cut rule

a rule for composing proofs

$$\dfrac{A \to B,\ \neg B \vdash \neg A \qquad\qquad B \to C,\ \neg C \vdash \neg B}{A \to B,\ B \to C,\ \neg C \vdash \neg A}$$

?

$$(A \to B) \to (A \to C) \vdash A \to (B \to C)$$

Natural Deduction

one natural way to prove $A \to B$ is to assume A and prove B

a proof

$$\dfrac{A\to B \qquad \dfrac{B\to C \qquad \neg C}{\neg B}}{\neg A}$$

$\to$ introduction

a rule of inference

$$\dfrac{\Gamma, X \vdash Y}{\Gamma \vdash X \to Y}$$

a proof?

$$\dfrac{A\to B \qquad \dfrac{B\to C \qquad \cancel{\neg C}}{\neg B}}{\neg C \to \neg A}$$

$$\dfrac{\dfrac{A\to B,\ \neg B \vdash \neg A \qquad\qquad B\to C,\ \neg C \vdash \neg B}{A\to B,\ B\to C,\ \neg C \vdash \neg A}}{A\to B,\ B\to C \vdash \neg C \to \neg A}$$

Natural Deduction

one natural way to prove X→Y is to assume X and prove Y
and if we can prove X→Y then from X we can infer Y

→ introduction & elimination

a 2-way rule of inference

$$\begin{array}{c} \Gamma, X \vdash Y \\ \hline\hline \Gamma \vdash X \to Y \end{array}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{A,B \vdash B}{B \vdash A\to B} \quad \dfrac{(A\to B)\to(A\to C) \vdash (A\to B)\to(A\to C)}{A\to B, (A\to B)\to(A\to C) \vdash A\to C}}{B, (A\to B)\to(A\to C) \vdash A\to C} \qquad \dfrac{A\to C \vdash A\to C}{A\to C, A \vdash C}}{(A\to B)\to(A\to C), A, B \vdash C}}{(A\to B)\to(A\to C), A \vdash B\to C}}{(A\to B)\to(A\to C) \vdash A \to (B\to C)}$$

The proofs may be natural, but sometimes they are hard to find!

Gentzen's idea

Instead of just entailments, $\Gamma \vdash X$

(where X is an expression and Γ is a finite set of expressions)

allow **sequents**, $\Gamma \vdash \Delta$

(where both Γ and Δ are finite sets of expressions)

Of course, every entailment 'is' a sequent

(where Δ is a singleton)

but the sequent calculus is much simpler than natural deduction

KEEP
CALM
&
FOLLOW
THE RULES

$$\frac{}{\Gamma, A \vdash \Delta, A}\ (I)$$

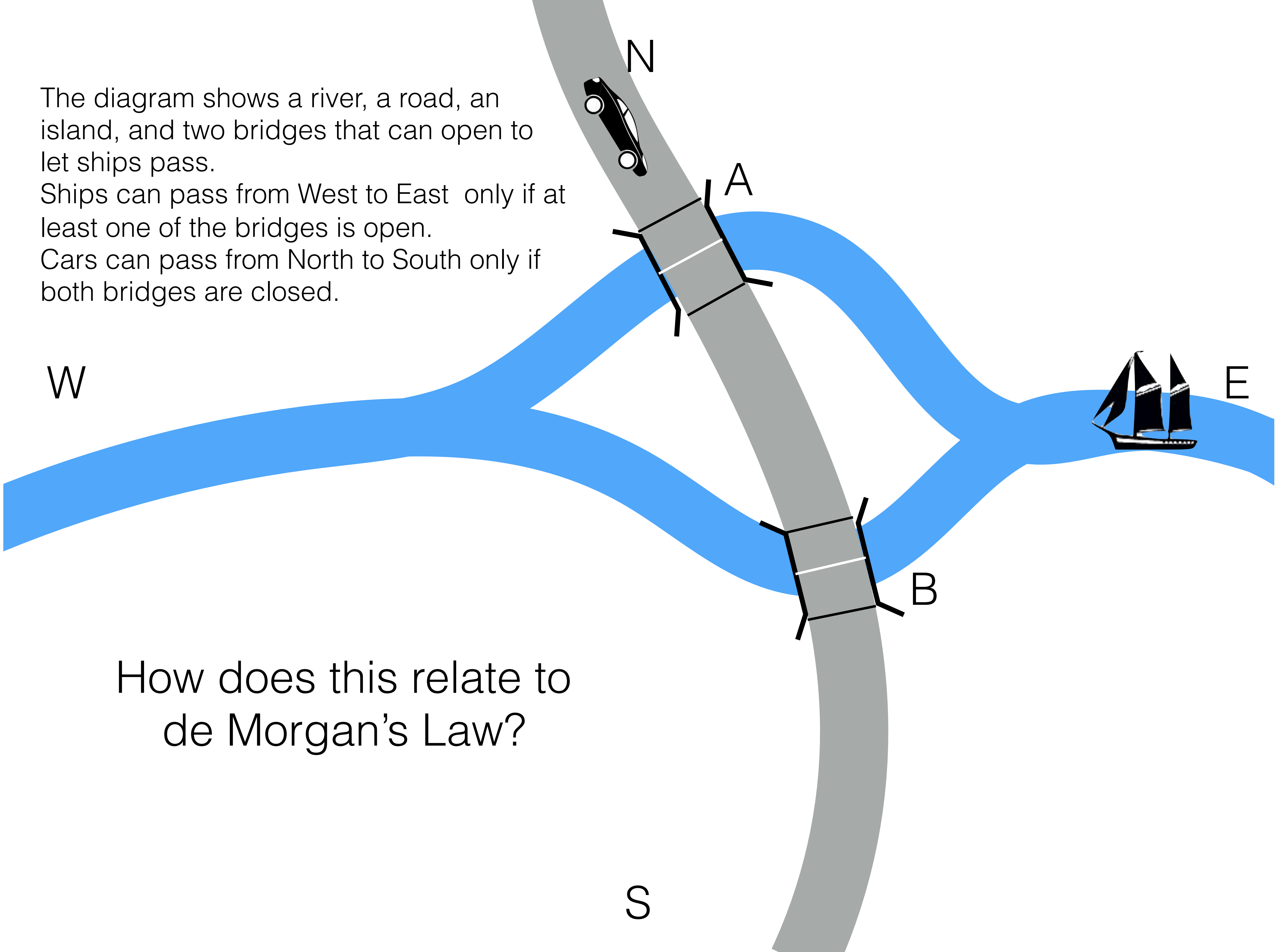
$$\frac{\Gamma, A, B \vdash \Delta}{\Gamma, A \wedge B \vdash \Delta}\ (\wedge L) \qquad \frac{\Gamma \vdash A, B, \Delta}{\Gamma \vdash A \vee B, \Delta}\ (\vee R)$$

$$\frac{\Gamma, A \vdash \Delta \quad \Gamma, B \vdash \Delta}{\Gamma, A \vee B \vdash \Delta}\ (\vee L) \qquad \frac{\Gamma \vdash A, \Delta \quad \Gamma \vdash B, \Delta}{\Gamma \vdash A \wedge B, \Delta}\ (\wedge R)$$

$$\frac{\Gamma \vdash A, \Delta \quad \Gamma, B \vdash \Delta}{\Gamma, A \to B \vdash \Delta}\ (\to L) \qquad \frac{\Gamma, A \vdash B, \Delta}{\Gamma \vdash A \to B, \Delta}\ (\to R)$$

$$\frac{\Gamma \vdash A, \Delta}{\Gamma, \neg A \vdash \Delta}\ (\neg L) \qquad \frac{\Gamma, A \vdash \Delta}{\Gamma \vdash \neg A, \Delta}\ (\neg R)$$

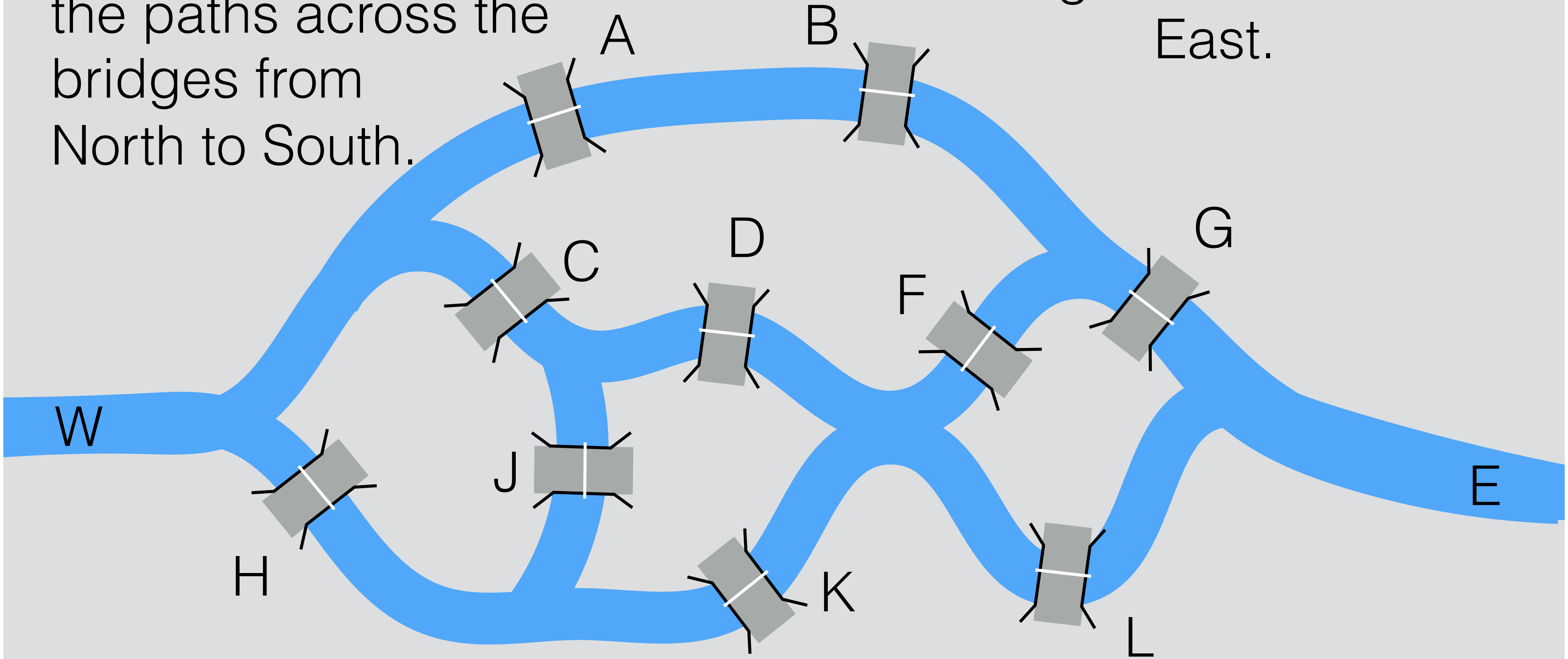
The diagram shows a river, a road, an island, and two bridges that can open to let ships pass.
Ships can pass from West to East only if at least one of the bridges is open.
Cars can pass from North to South only if both bridges are closed.



How does this relate to de Morgan's Law?

Draw a graph showing the paths across the bridges from North to South.

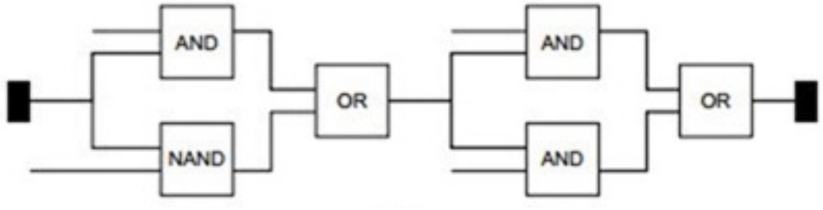
Draw a graph showing the paths under the bridges from West to East.

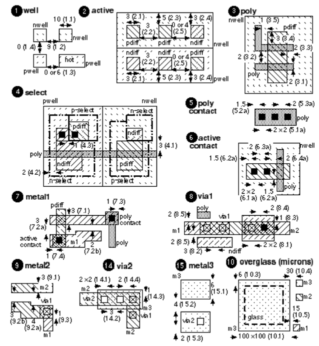


In each case, the bridges correspond to edges of the graph.

What is the logical relationship between the two graphs?

	7				6			
9							4	1
		8			9		5	
	9				7			2
		3				8		
4			8				1	
	8		3			9		
1	6							7
			5				8	





We can express many combinatorial problems in propositional logic

(eg Sudoku, but also more practical problems, such as circuit design)

We can use resolution to check whether a set of clauses is consistent.

If we can derive the empty clause the set is inconsistent, and we can invert the proof to produce a refutation tree

If we cannot derive the empty clause we can construct a satisfying valuation from the failed attempt to prove a contradiction

Generating all the resolvants takes space and time

Davis Putnam

Take a collection $\mathcal{C}$ of clauses.

For each propositional letter, A

For each pair $(X, Y) \mid X \in \mathcal{C} \wedge Y \in \mathcal{C} \wedge \mathbf{A} \in X \wedge \neg\mathbf{A} \in Y$

if $\mathcal{R}(X, Y, \mathbf{A}) = \{\}$ return UNSAT

if $\mathcal{R}(X, Y, \mathbf{A})$ is consistent $\mathcal{C} := \mathcal{C} \cup \{ \mathcal{R}(X, Y) \}$

return SAT

Where $\mathcal{R}(X, Y, \mathbf{A}) = X \cup Y \setminus \{ \mathbf{A}, \neg\mathbf{A} \}$

Heuristic: start with variables that occur seldom.

Naïve search

V is a partial valuation
(a consistent set of literals)
$V \wedge A = V \cup \{A\}$

```
function SAT(Φ,V)
  Φ|V = {}
  ||
    {} ∉ Φ|V
    &&
    let A = chooseLiteral (Φ,V)
    in
       SAT (Φ,V ^ A)
       ||
       SAT (Φ,V ^ ¬A)
```

Φ is a set of clauses

$\Phi \mid V$ is the result of simplifying Φ using V:
For each literal $L \in V$

- remove clauses containing L
- delete $\neg L$ from remaining clauses

chooseLiteral(Φ,V) returns a literal occurring in $\Phi \mid V$

We can express many combinatorial problems in propositional logic

(eg Sudoku, but also more practical problems)

We can search for solutions to a set of constraints expressed in propositional logic

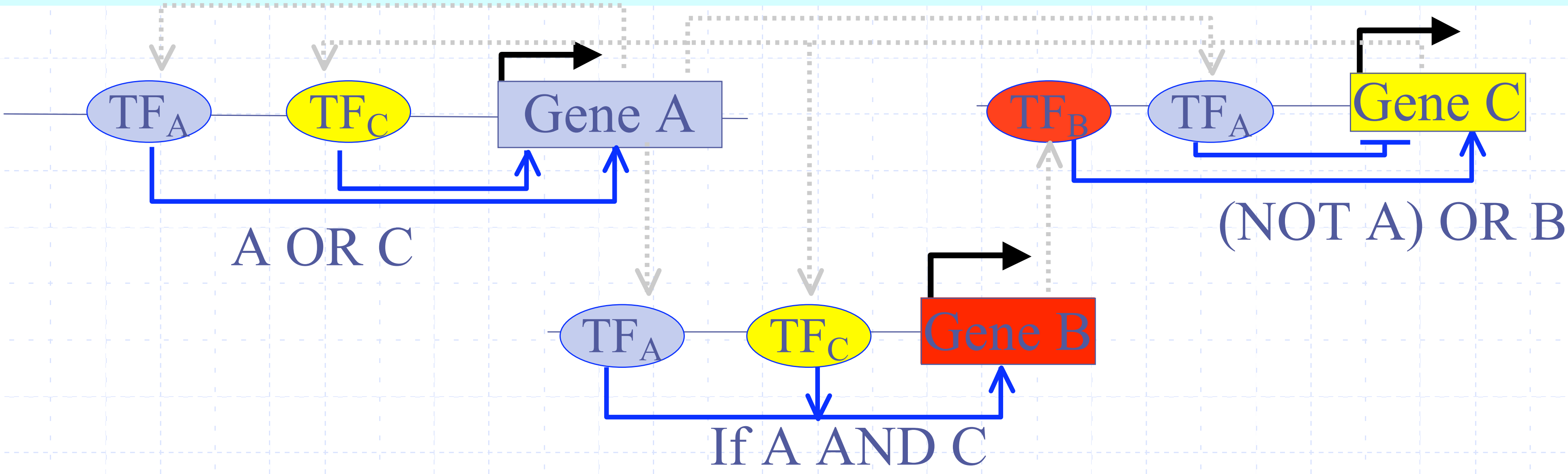
We convert the problem to clausal form
and check partial valuations V against our constraints
if V contradicts any clauses our search must backtrack.

Checking these potential solutions costs time and space

We can narrow the search by **unit propagation**:
identifying literals whose siblings are all falsified,
and making them true

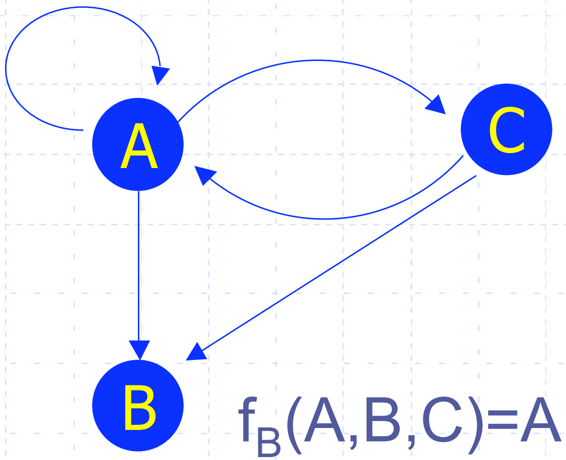
Keeping track of unit literals takes time

Boolean Network Model



- A Boolean Network Model:
  - Nodes represent transcription factors
  - Edges represent regulatory input
  - Boolean gates (input functions) represent gene expression

$f_A(A,B,C)$=A OR C



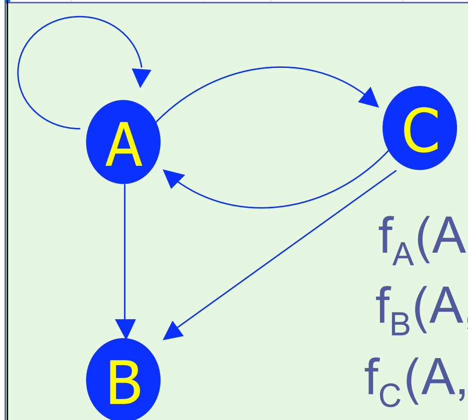
$f_C(A,B,C)$=NOT A OR B)

$f_B(A,B,C)$=A AND C

Dynamics

- Network State: X=(A,B,C…) is a Boolean vector
- State evolution: $X(t+1)=f(X(t))=(f_A(X(t)),f_B(X(t)), \ldots.)$
  - E.g., X(t+1)=(A OR C, A AND C, (NOT A) OR B)
  - (0,1,1)=>(1,0,1)
- This is discrete time synchronous dynamics
  - State transitions occur through concurrent gates firings

X(t)	X(t+1)
000	001
001	101
010	001
011	101
100	100
101	110
110	101
111	111

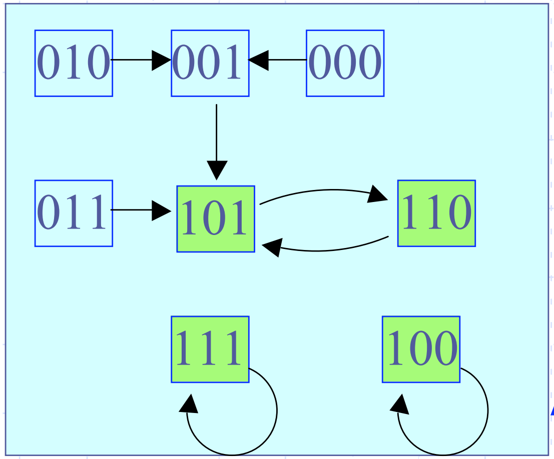


$f_A(A,B,C)$=A OR C

$f_B(A,B,C)$=A AND C

$f_C(A,B,C)$=(NOT A) OR B

State-space dynamics



Cycle

Attractors

Regular expressions and finite automata

Let A be a class of such strings. We call A <u>regular</u>, if A can be described by an expression built out of the following operations (chosen in analogy to the definition of regular events in Sect. 7.1.)

The empty set and the unit set consisting of just $a_{\underline{i}}$ for any $\underline{i}$ are <u>regular</u>. If A and B are regular, so is their sum which we write $A \vee B$. If A and B are regular, so is the set, written AB, of strings obtained by writing a string belonging to A just left of a string belonging to B. If A and B are regular, so is A*B which abbreviates $\overbrace{A \cdots A}^{\underline{n}\text{ factors}}B$ $(\underline{n} \geq 0)$, i.e., the sum of these classes for all $\underline{n} \geq 0$.

S. C. Kleene 1909–1994

Representation of events in nerve nets and finite automata 1951

https://www.rand.org/content/dam/rand/pubs/research_memoranda/2008/RM704.pdf

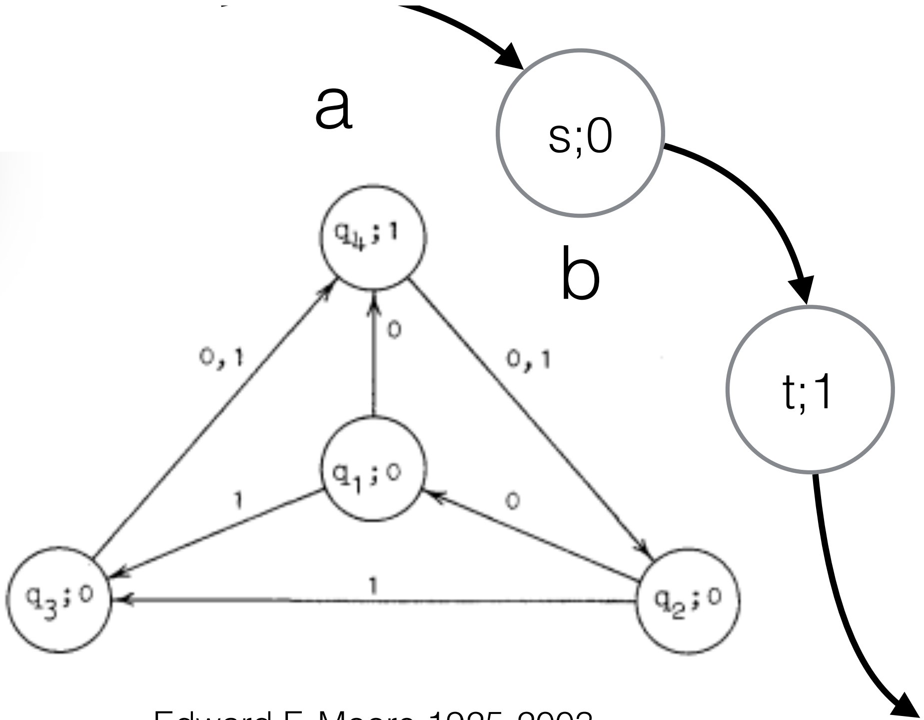
Moore machine

Mealy machine

Previous State	Present State: Previous Input 0	Present State: Previous Input 1
q_1	q_4	q_3
q_2	q_1	q_3
q_3	q_4	q_4
q_4	q_2	q_2

Present State	Present Output
q_1	0
q_2	0
q_3	0
q_4	1

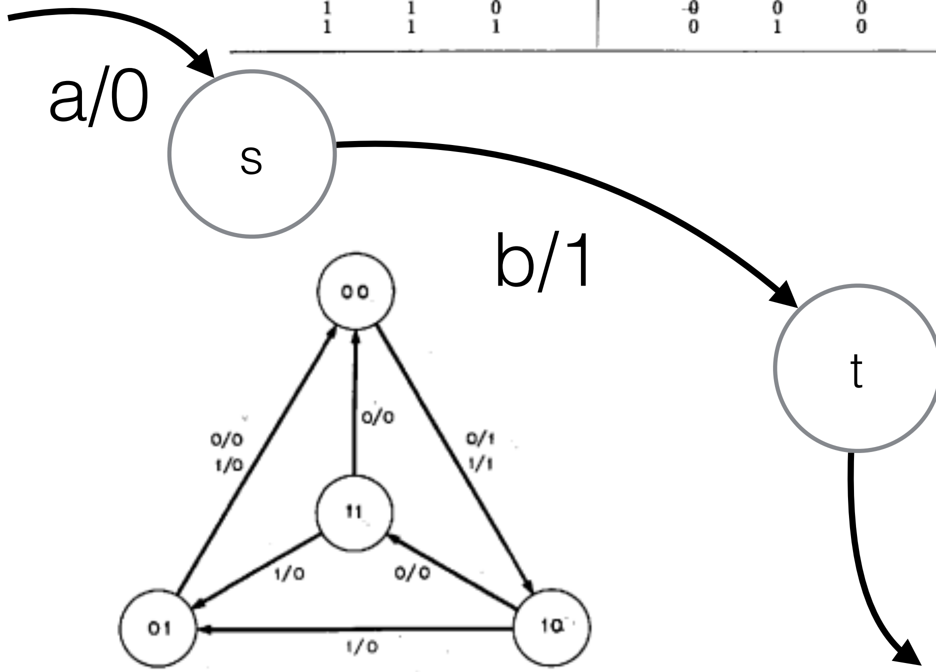
q_1	q_2	x	$\bar{q}_1$	$\bar{q}_2$	y
0	0	0	1	0	1
0	0	1	1	0	1
0	1	0	0	0	0
0	1	1	0	0	0
1	0	0	1	1	0
1	0	1	0	1	0
1	1	0	0	0	0
1	1	1	0	1	0



Edward F. Moore 1925-2003

Gedanken - Experiments on Sequential Machines, 1956

http://people.mokk.bme.hu/~kornai/termeszetes/moore_1956.pdf



George H. Mealy 1927-2010

A Method for
Synthesizing Sequential Circuits

http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6771467

Finite State Machine concepts proved valuable in language parsing (compilers) and sequential circuit design

Moore is less

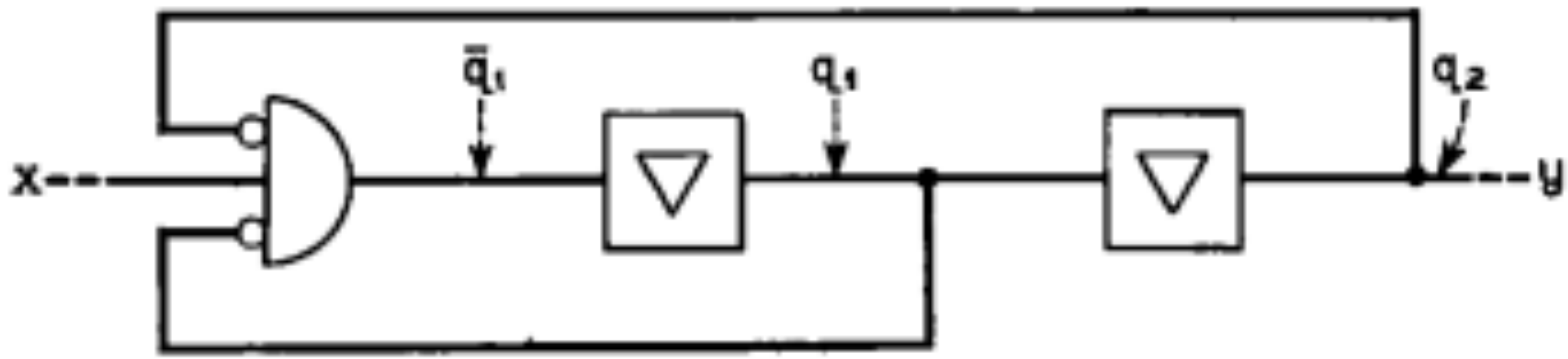
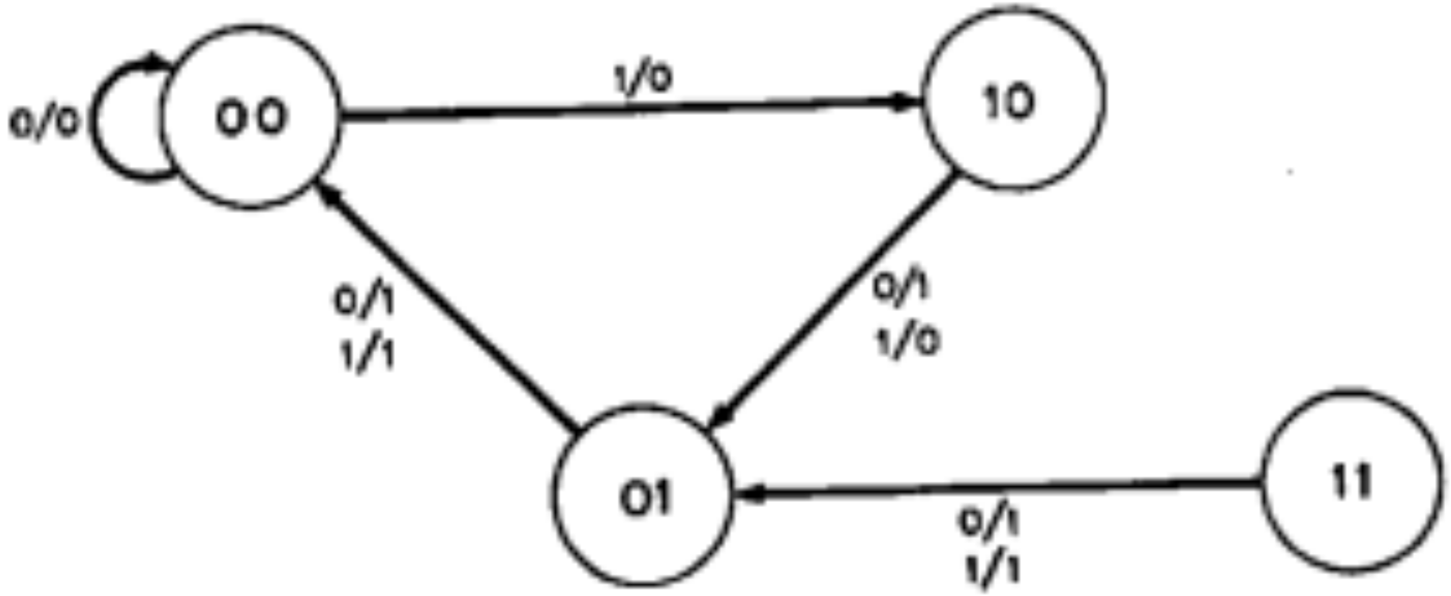


Fig. 5



A nondeterministic automaton has, at each stage of its operation, several choices of possible actions. This versatility enables us to construct very powerful automata using only a small number of internal states.

Nondeterministic automata, however, turn out to be equivalent to the usual automata. This fact is utilized for showing quickly that certain sets are definable by automata.

Dana S. Scott 1934-…

Michael O. Rabin 1931-…

Finite Automata and their Decision Problems 1959

http://www.cse.chalmers.se/~coquand/AUTOMATA/rs.pdf

Finite State Machine Parsing for Internet Protocols: Faster Than You Think (2014)

Parsers are responsible for translating unstructured, untrusted, opaque data to a structured, implicitly trusted, semantically meaningful format suitable for computing on. Parsers, therefore, are the components that facilitate the separation of data from computation and, hence, exist in nearly every conceivable useful computer system

Parsers must be correct, so that only valid input is blessed with trust; and they must be efficient so that enormous documents and torrential datastreams don't bring systems to their knees

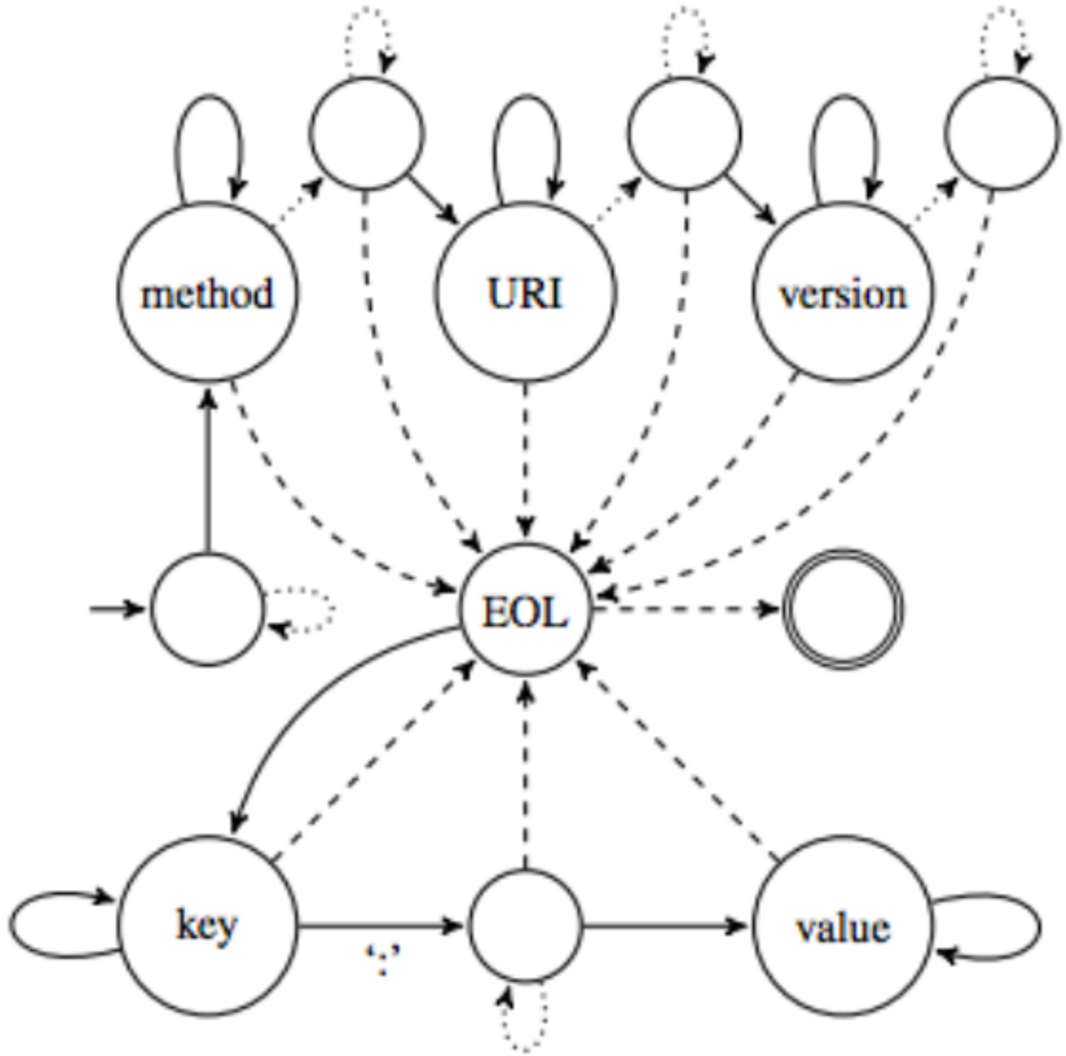


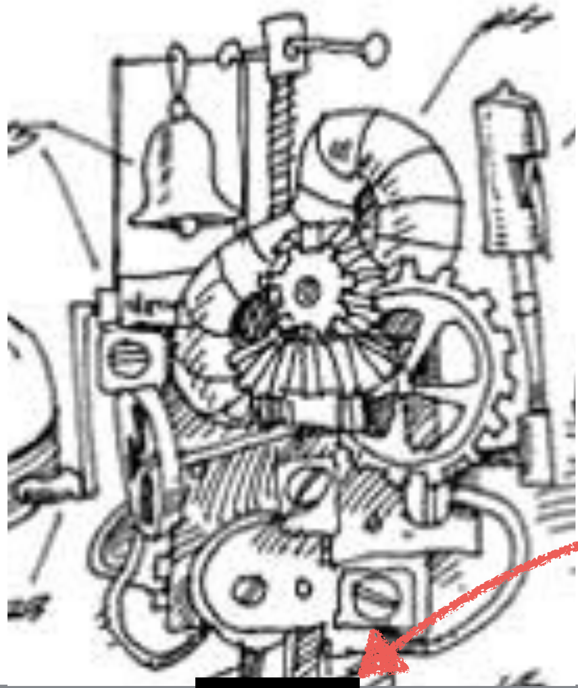
Fig. 2. DFA that recognizes HTTP headers, with the request on the first line, followed by arbitrary key-value pairs on subsequent lines, ending with two consecutive newlines. Solid edges represent any printable character, dotted lines represent a space, dashed lines represent a newline. The unlabeled states just eat spaces.

A Practical Introduction to Hardware/Software Codesign,
Chapter 4. Finite State Machine with Datapath (2010)

Abstract In this chapter, we introduce an important building block for efficient custom hardware design: the Finite State Machine with Datapath (FSMD). An FSMD combines a controller, modeled as a finite state machine (FSM) and a datapath. The datapath receives commands from the controller and performs operations as a result of executing those commands. The controller uses the results of data path operations to make decisions and to steer control flow. The FSMD model will be used throughout the remainder of the book as the reference model for the 'hardware' part of hardware/software codesign.

FSM

Moore



+1 →

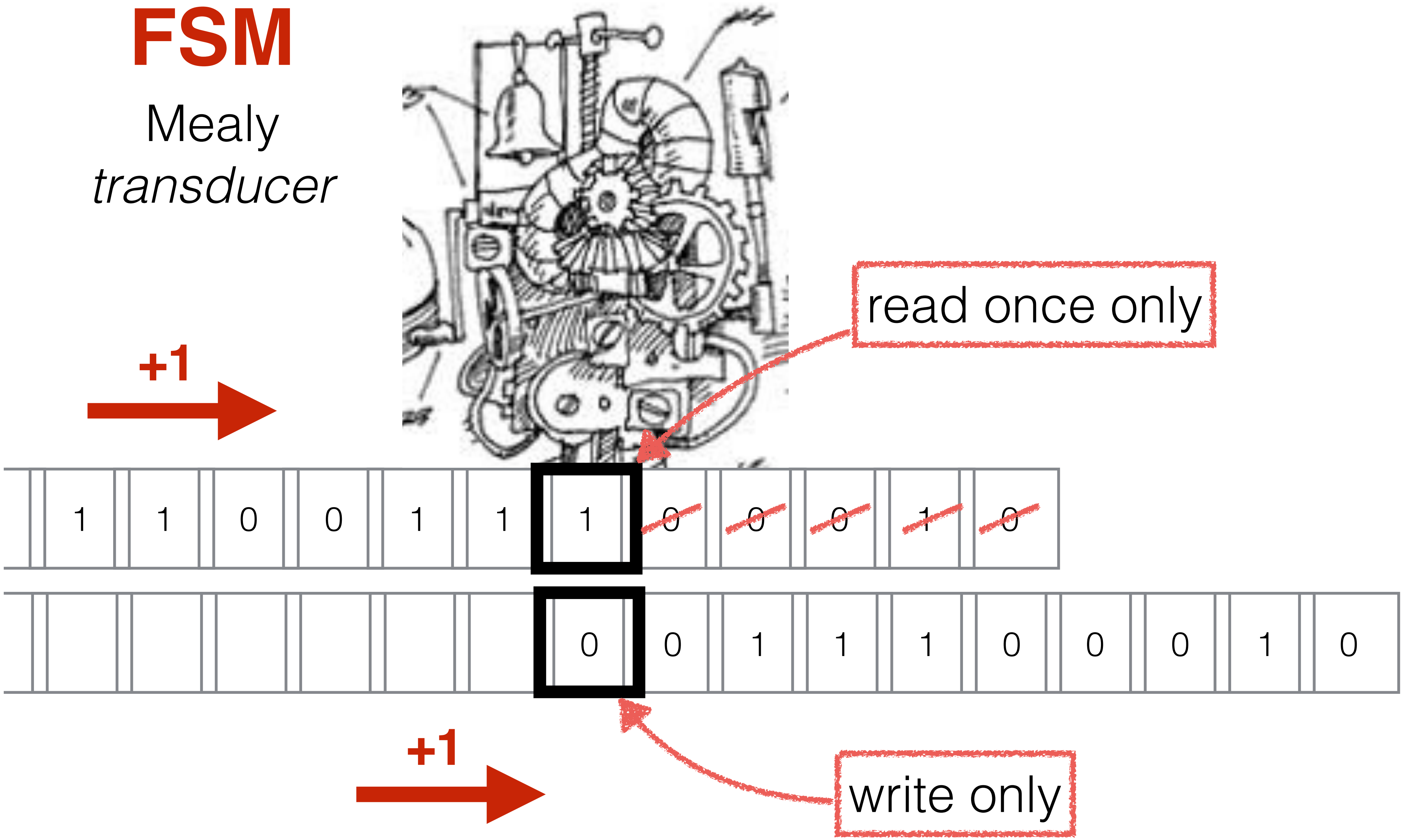
0	1	1	0	0	1	1	$\boxed{1}$	~~0~~	~~0~~	~~0~~	~~1~~	~~0~~

read once only

FSM

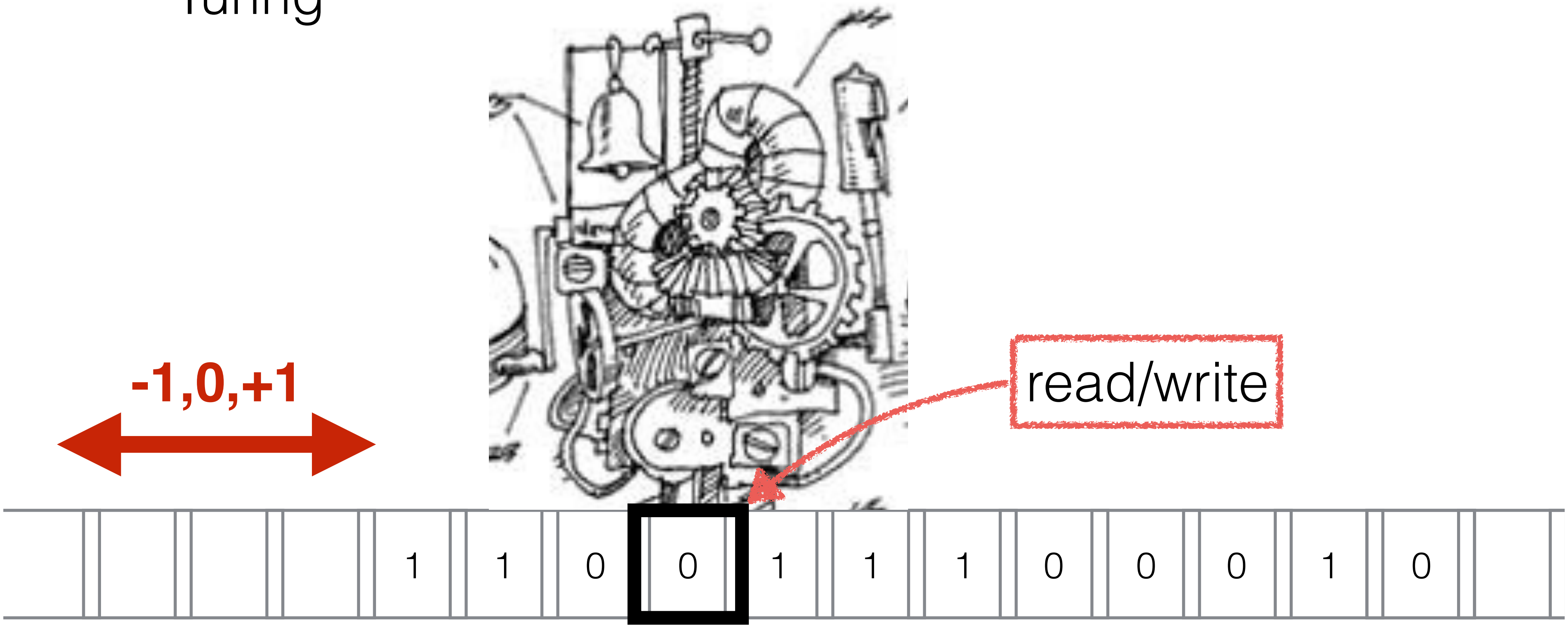
Mealy

transducer

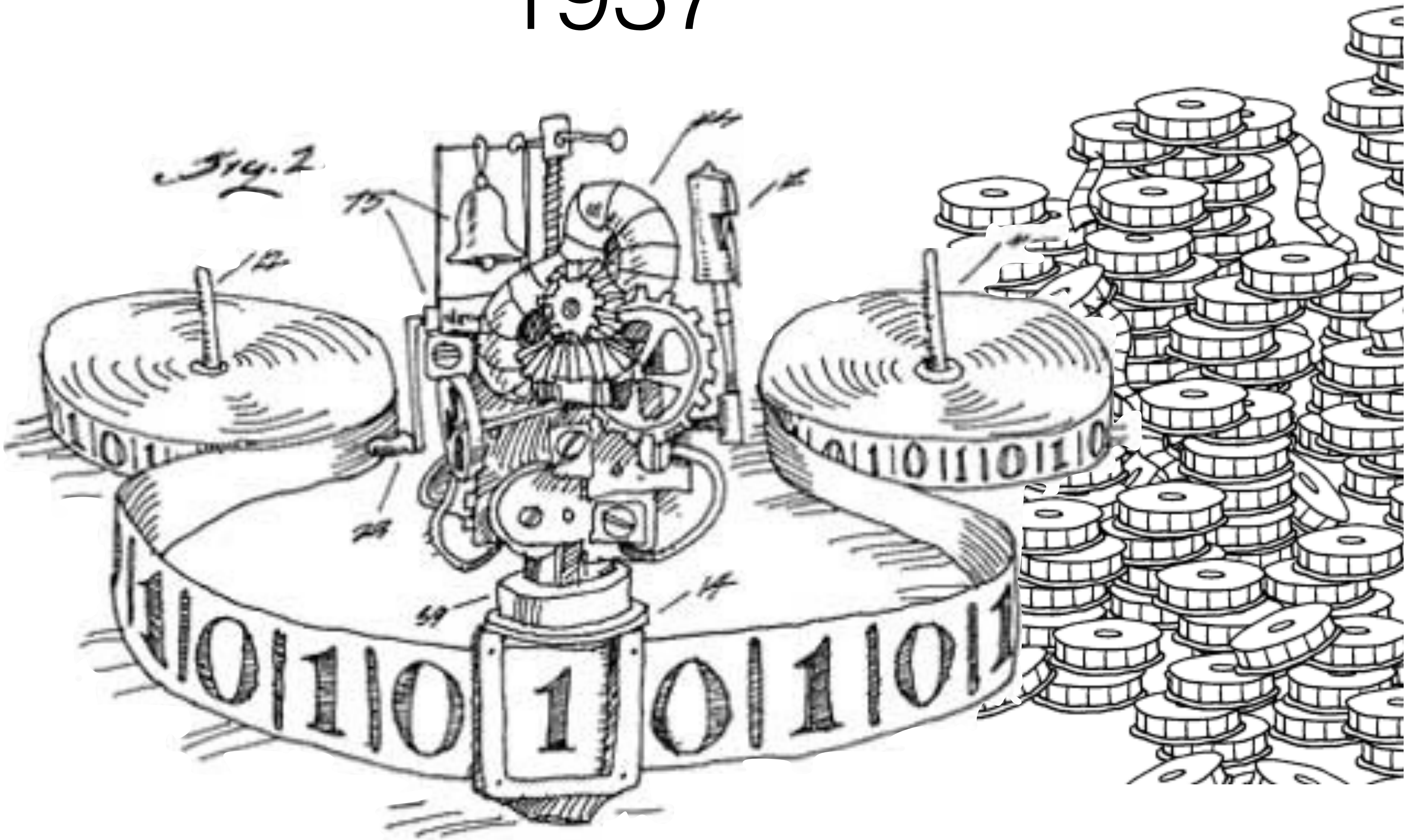


Turing Machine

Turing



Universal Turing Machine
1937



Alan Turing 1912-1954