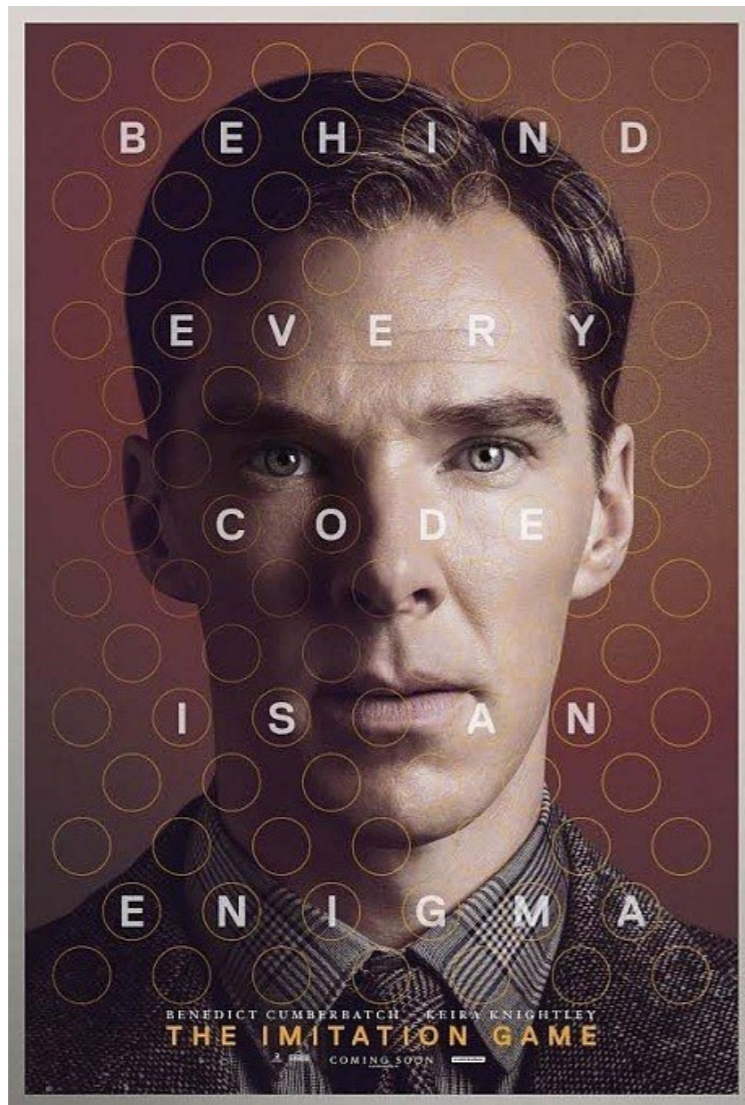


Automata & languages

A primer on the Theory of Computation



Laurent Vanbever

nsg.ee.ethz.ch

ETH Zürich (D-ITET)

October 15 2020

Part 5 out of 5

Last week was all about

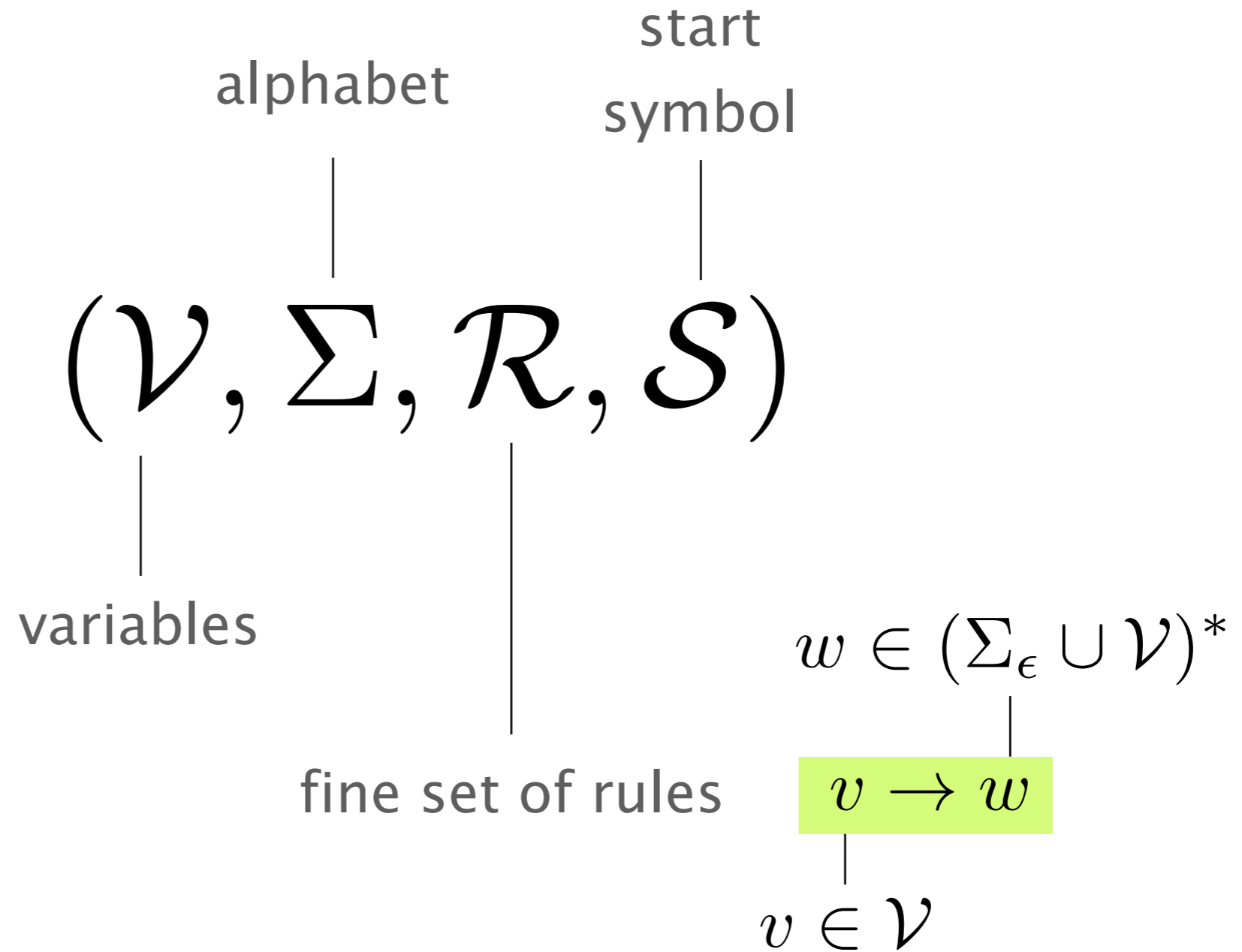
Context-Free Languages

Context-Free Languages

a superset of Regular Languages

Example $\{0^n 1^n \mid n \geq 0\}$ is a CFL but not a RL

We saw the concept of Context-Free Grammars



As for Regular Languages,
Context-Free Languages are recognized by “machines”

Language

Regular

Context-Free

Machine

DFA/NFA

PDA

Push-Down Automatas are pretty similar to DFAs

$$M = (Q, \Sigma, \Gamma, \delta, q_0, F)$$

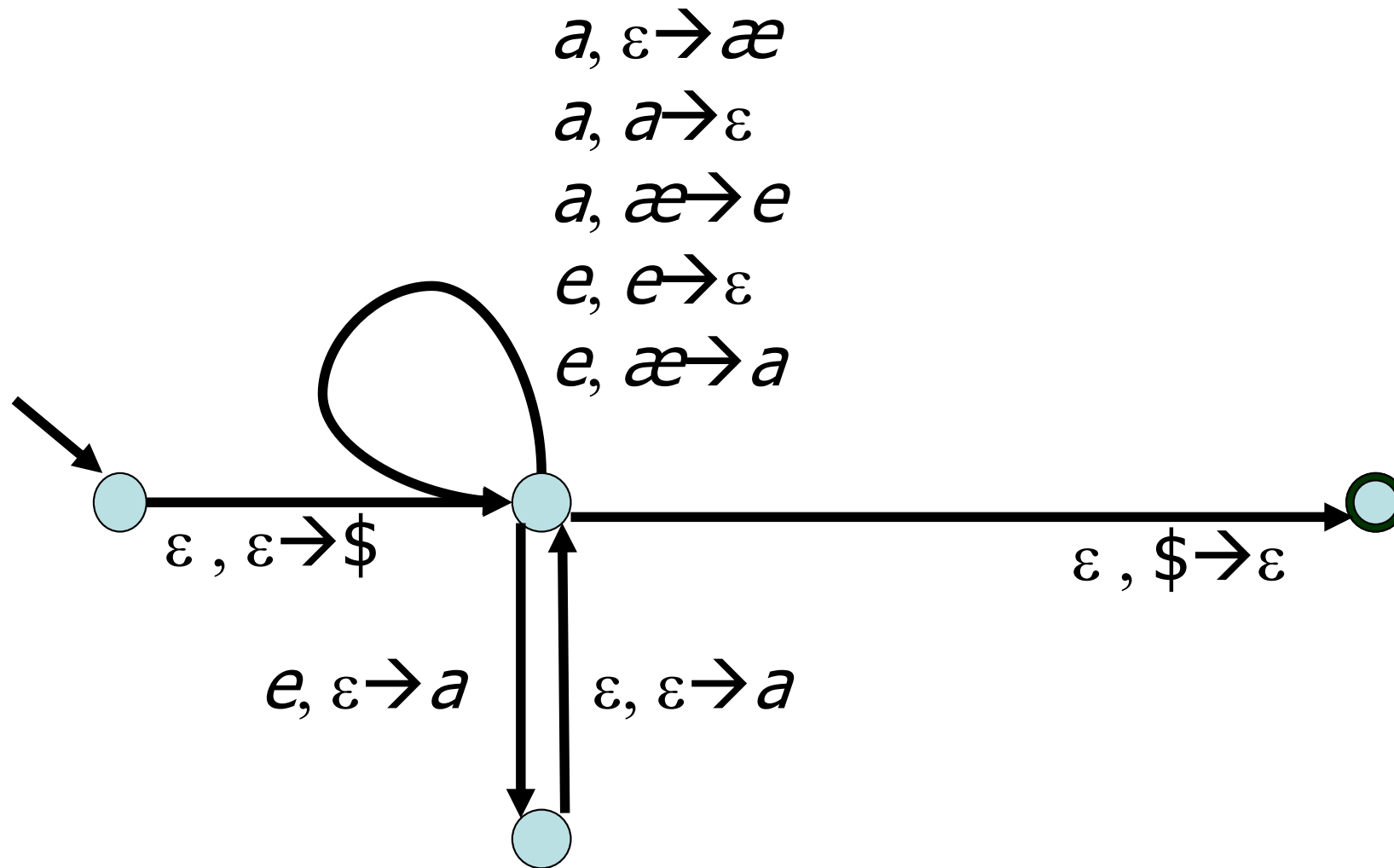
Diagram illustrating the components of a Push-Down Automaton (PDA) M and their corresponding labels:

- Q : states
- Σ : alphabet
- Γ : start state
- δ : transition function
- q_0 : start state
- F : accepting states

Solution 2

- The idea is to use the stack to keep count of the number of a 's and/or e 's needed to get a valid string. If we have a surplus of e 's thus far, we should have corresponding number of a 's (two for every e) on the stack. On the other hand, if we have a surplus of a 's we *cannot* put e 's on the stack since we can't split symbols. So instead, we use a new symbol æ , which corresponds to a needed pair of a and e .
- Let's see how this looks on the next slide.

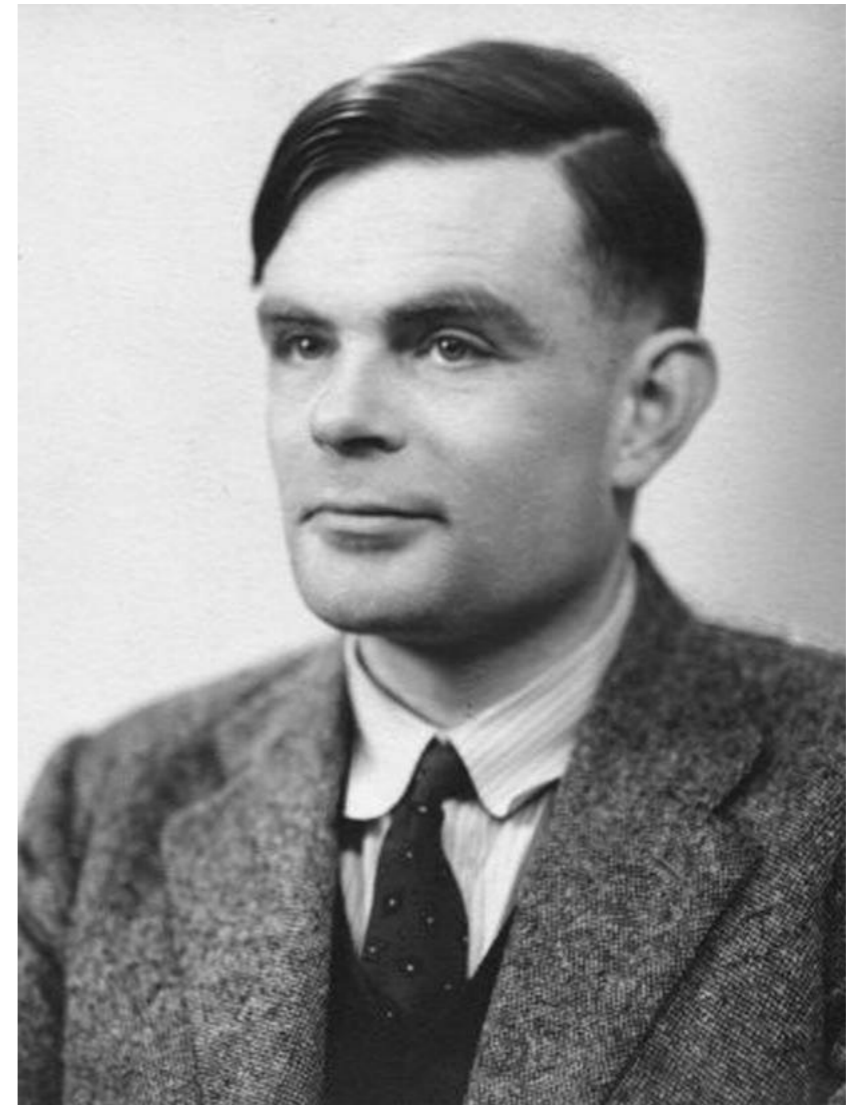
Solution 2 (nondeterministic)



This week, we'll see that
computers are not limitless

Some problems
cannot be solved
by a computer
(no matter its power)

Alan Turing (1912-1954)



But before that, we'll prove
some extra properties about Context-Free Languages

Today's plan

Thu Oct 15

- 1 PDA \approx CFG
- 2 Pumping lemma for CFL
- 3 Turing Machines

Even smarter automata...

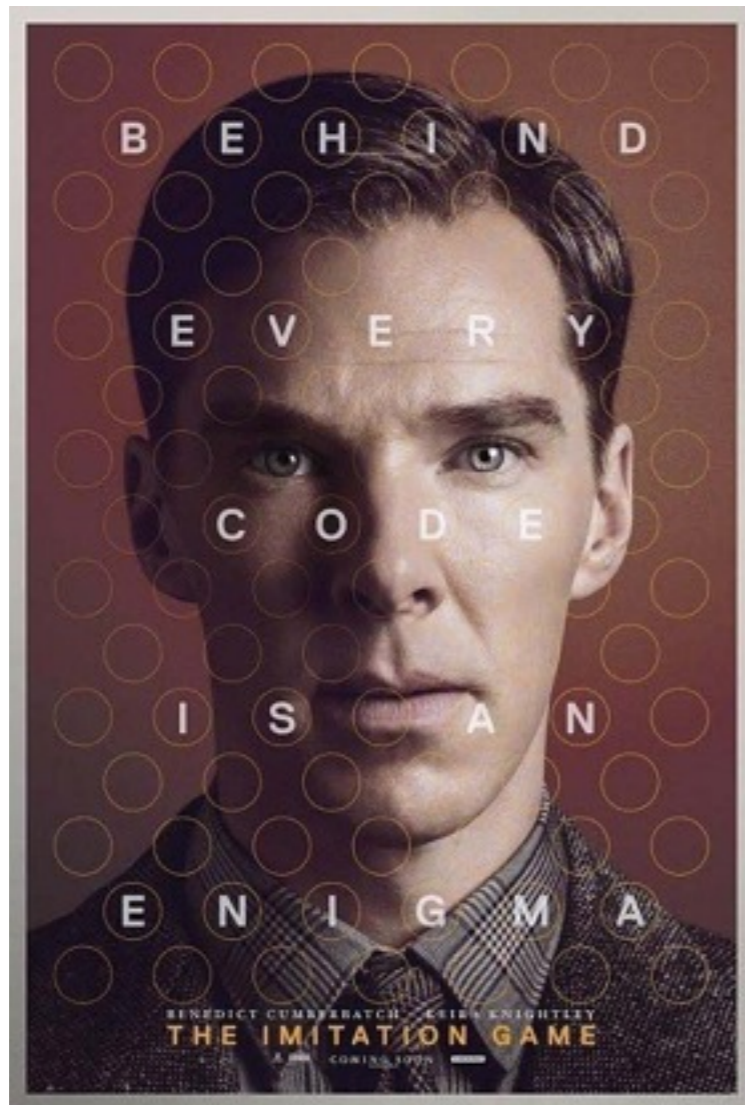
- Even though the PDA is more powerful than the FA, it is still **really** stupid, since it doesn't understand a lot of important languages.
- Let's try to make it more powerful by adding a **second stack**
 - You can push or pop from either stack, also there's still an input string
 - Clearly there are quite a few "implementation details"
 - It seems at first that it doesn't help a lot to add a second stack, but...

Even smarter automata...

- Even though the PDA is more powerful than the FA, it is still **really** stupid, since it doesn't understand a lot of important languages.
- Let's try to make it more powerful by adding a **second stack**
 - You can push or pop from either stack, also there's still an input string
 - Clearly there are quite a few "implementation details"
 - It seems at first that it doesn't help a lot to add a second stack, but...
- Lemma: A PDA with two stacks is **as powerful as** a machine which operates on an infinite tape (restricted to read/write only "current" tape cell at the time – known as "Turing Machine").
 - Still that doesn't sound very exciting, does it...?!?

Automata & languages

A primer on the Theory of Computation



regular
language

context-free
language

Part 3

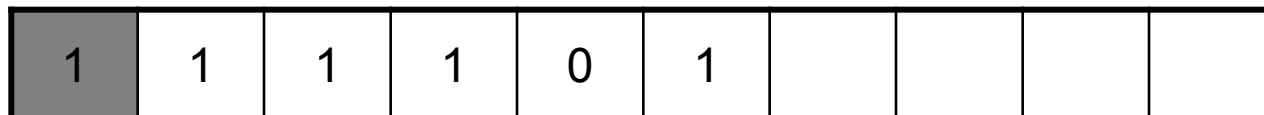
turing
machine

Turing Machine

- A **Turing Machine (TM)** is a device with a finite amount of *read-only* “*hard*” memory (states), and an unbounded amount of read/write tape-memory. There is no separate input. Rather, the input is assumed to reside on the tape at the time when the TM starts running.
- Just as with Automata, TM’s can either be input/output machines (compare with Finite State Transducers), or yes/no decision machines.

Turing Machine: Example Program

- Sample Rules:
 - If read 1, write 0, go right, repeat.
 - If read 0, write 1, HALT!
 - If read \square , write 1, HALT! (the symbol \square stands for the blank cell)
- Let's see how these rules are carried out on an input with the *reverse* binary representation of 47:



Turing Machine: Formal Definition

- Definition: A **Turing machine** (TM) consists of a 7-tuple

$$M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{acc}}, q_{\text{rej}}).$$

- Q , Σ , and q_0 , are the same as for an FA.
- q_{acc} and q_{rej} are accept and reject states, respectively.
- Γ is the tape alphabet which necessarily contains the blank symbol \bullet , as well as the input alphabet Σ .
- δ is as follows:

$$\delta : (Q - \{q_{\text{acc}}, q_{\text{rej}}\}) \times \Gamma \rightarrow Q \times \Gamma \times \{\text{L}, \text{R}\}$$

- Therefore given a non-halt state p , and a tape symbol x , $\delta(p, x) = (q, y, D)$ means that TM goes into state q , replaces x by y , and the tape head moves in direction D (left or right).

Turing Machine: Formal Definition

- Definition: A **Turing machine** (TM) consists of a 7-tuple

$$M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{acc}}, q_{\text{rej}}).$$

- Q , Σ , and q_0 , are the same as for an FA.
- q_{acc} and q_{rej} are accept and reject states, respectively.
- Γ is the tape alphabet which necessarily contains the blank symbol \bullet , as well as the input alphabet Σ .
- δ is as follows:

$$\delta : (Q - \{q_{\text{acc}}, q_{\text{rej}}\}) \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$$

- Therefore given a non-halt state p , and a tape symbol x , $\delta(p, x) = (q, y, D)$ means that TM goes into state q , replaces x by y , and the tape head moves in direction D (left or right).
- A string x is **accepted** by M if after being put on the tape with the Turing machine head set to the left-most position, and letting M run, M eventually enters the accept state. In this case w is an element of $L(M)$
 - the language accepted by M .

Comparison

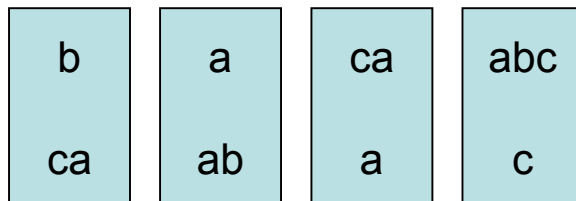
Device	Separate Input?	Read/Write Data Structure	Deterministic by default?
FA	Yes	None	Yes
PDA	Yes	LIFO Stack	No
TM	No	1-way infinite tape. 1 cell access per step.	Yes (but will also allow crashes)

Turing Machine: Goals

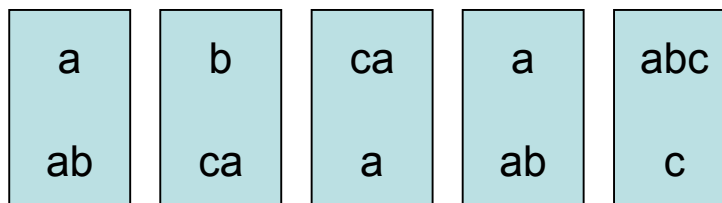
- First Goal of Turing's Machine: A "computer" which is as **powerful** as any real computer/programming language
 - As powerful as C, or "Java++"
 - Can execute all the same algorithms / code
 - Not as fast though (move the head left and right instead of RAM)
 - Historically: A model that can compute anything that a human can compute. Before invention of electronic computers the term "computer" actually referred to a *person* who's line of work is to calculate numerical quantities!
 - This is known as the [Church-[Post-]] Turing thesis, 1936.
- Second Goal of Turing's Machine: And at the same time a model that is **simple** enough to actually prove interesting epistemological results.

Can a computer compute anything...?!?

- Given collection of dominos, e.g.



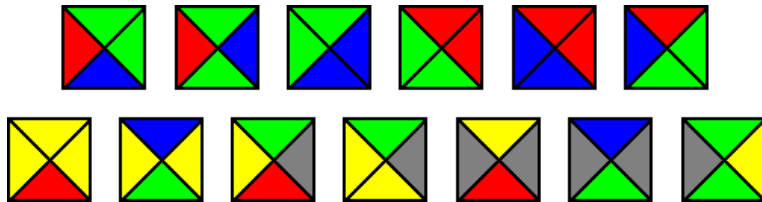
- Can you make a list of these dominos (repetitions are allowed) so that the top string equals the bottom string, e.g.



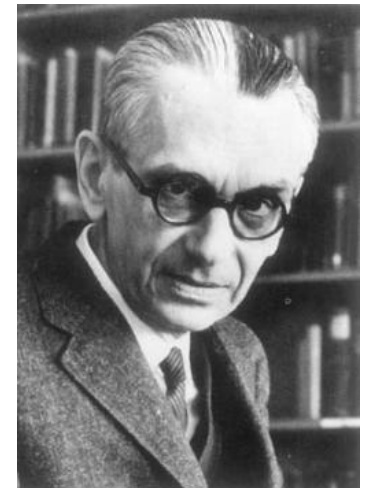
- This problem is known as Post-Correspondance-Problem.
- It is provably **unsolvable** by computers!

Also the Turing Machine (the Computer) is limited

- Similarly it is undecidable whether you can cover a floor with a given set of floor tiles (famous examples are Penrose tiles or Wang tiles)



- Examples are leading back to Kurt Gödel's incompleteness theorem
 - “Any powerful enough axiomatic system will allow for propositions that are undecidable.”

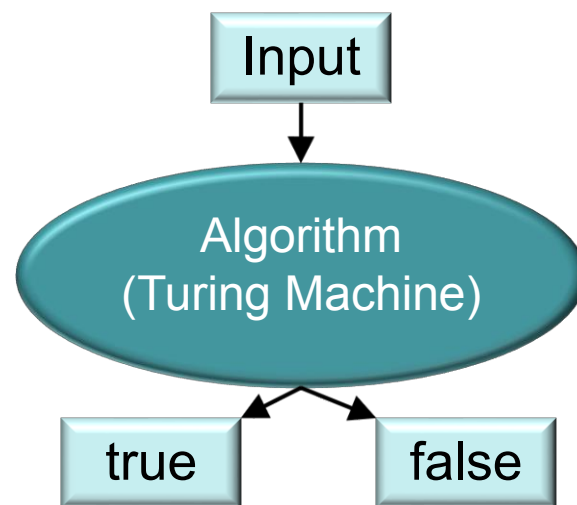


Decidability

- A **function is computable** if there is an algorithm (according to the Church-Turing-Thesis a **Turing machine** is sufficient) that computes the function (in finite time).

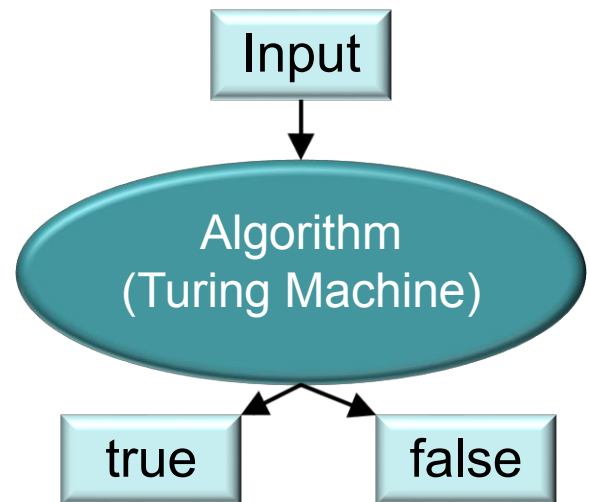
Decidability

- A **function is computable** if there is an algorithm (according to the Church-Turing-Thesis a **Turing machine** is sufficient) that computes the function (in finite time).
- A subset T of a set M is called **decidable** (or recursive), if the function $f: M \rightarrow \{\text{true}, \text{false}\}$ with $f(m) = \text{true}$ if $m \in T$, is **computable**.



Decidability

- A **function is computable** if there is an algorithm (according to the Church-Turing-Thesis a **Turing machine** is sufficient) that computes the function (in finite time).
- A subset T of a set M is called **decidable** (or recursive), if the function $f: M \rightarrow \{\text{true}, \text{false}\}$ with $f(m) = \text{true}$ if $m \in T$, is **computable**.
- A more general class are the **semi-decidable** problems, for which the algorithm must only terminate in finite time in either the true or the false branch, but not the other.



Halting Problem

- The halting problem is a famous example of an **undecidable** (semi-decidable) **problem**. Essentially, you cannot write a computer program that decides whether another computer program ever terminates (or has an infinite loop) on some given input.
- In pseudo code, we would like to have:

```
procedure halting(program, input) {  
    if program(input) terminates  
    then return true  
    else return false  
}
```

Halting Problem: Proof

- Now we write a little wrapper around our halting procedure

```
procedure test(program) {  
    if halting(program,program) =true  
    then loop forever  
    else return  
}
```

- Now we simply run: `test(test)!` **Does it halt?!?**

Excursion: P and NP

- **P** is the complexity class containing decision problems which can be solved by a Turing machine in time polynomial of the input size.

Excursion: P and NP

- **P** is the complexity class containing decision problems which can be solved by a Turing machine in time polynomial of the input size.
- **NP** is the class of decision problems solvable by a non-deterministic polynomial time Turing machine such that the machine answers "yes," if at least one computation path accepts, and answers "no," if all computation paths reject.

Excursion: P and NP

- **P** is the complexity class containing decision problems which can be solved by a Turing machine in time polynomial of the input size.
- **NP** is the class of decision problems solvable by a non-deterministic polynomial time Turing machine such that the machine answers "yes," if at least one computation path accepts, and answers "no," if all computation paths reject.
 - Informally, there is a Turing machine which can check the correctness of an answer in polynomial time.
 - E.g. one can check in polynomial time whether a traveling salesperson path connects n cities with less than a total distance d .

NP-complete problems

- An important notion in this context is the large set of **NP-complete** decision problems, which is a subset of NP and might be informally described as the "hardest" problems in NP.
- If there is a polynomial-time algorithm for even one of them, then there is a polynomial-time algorithm for **all** the problems in NP.
 - E.g. Given a set of n integers, is there a non-empty subset which sums up to 0? This problem was shown to be NP-complete.
 - Also the traveling salesperson problem is NP-complete, or Tetris, or Minesweeper.

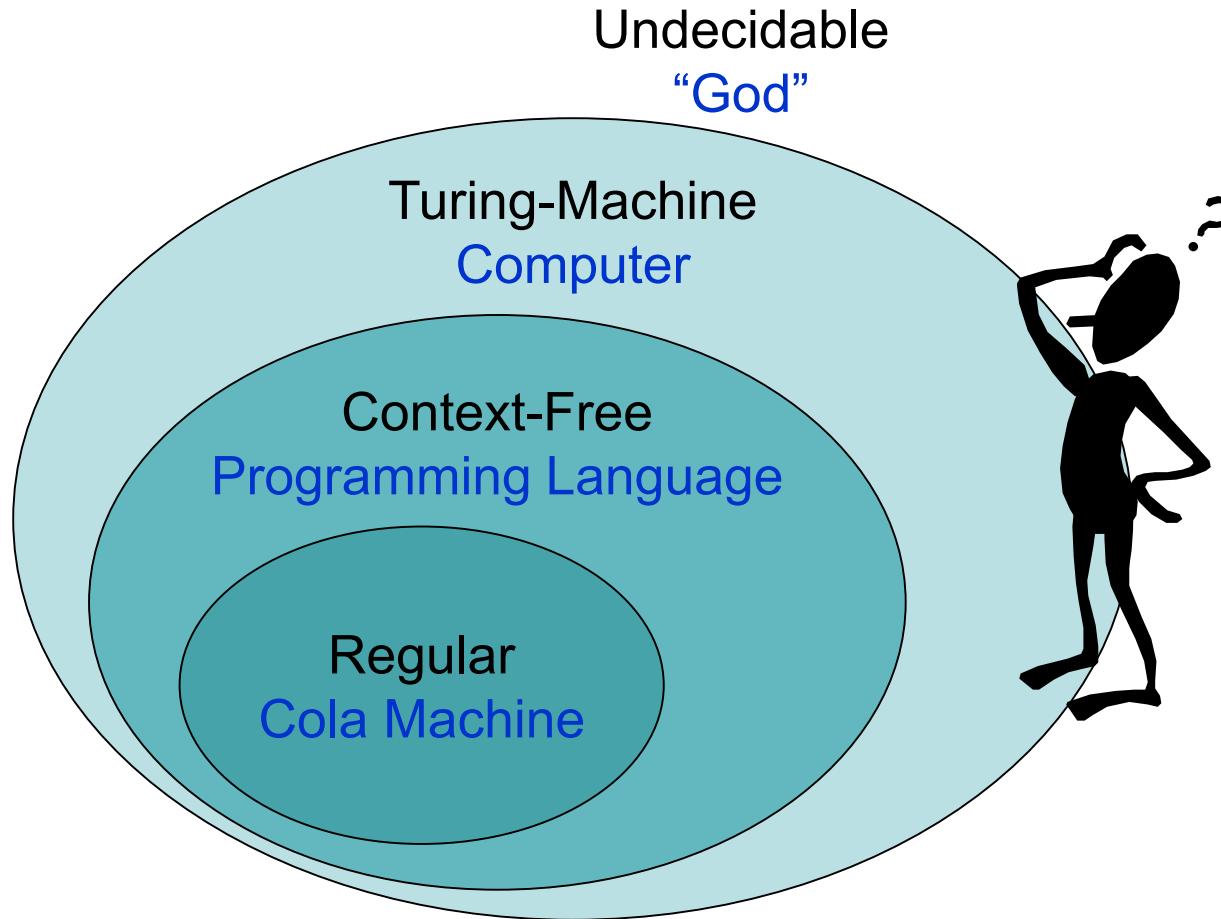
P = NP?

A glowing neon sign in the shape of the mathematical expression "P = NP?". The sign is made of bright white neon tubing and is mounted on a dark, vertically-grained wooden surface. The characters are stylized: the 'P' is a simple loop, the '=' consists of two parallel horizontal lines, the 'N' is a tall, narrow shape, the 'P' is another loop, and the '?' is a standard question mark. Two thin white wires are visible at the bottom, one on the left and one on the right, connected to the sign's terminals.

P vs. NP

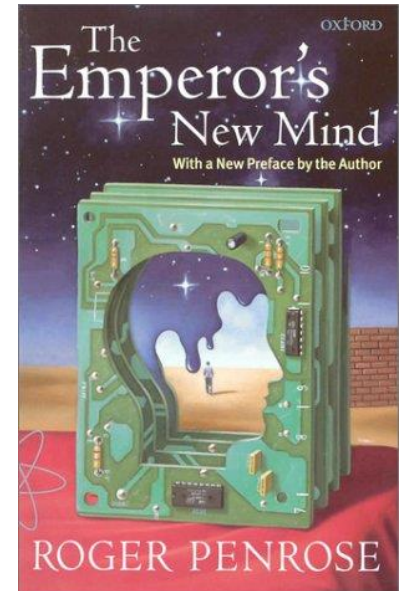
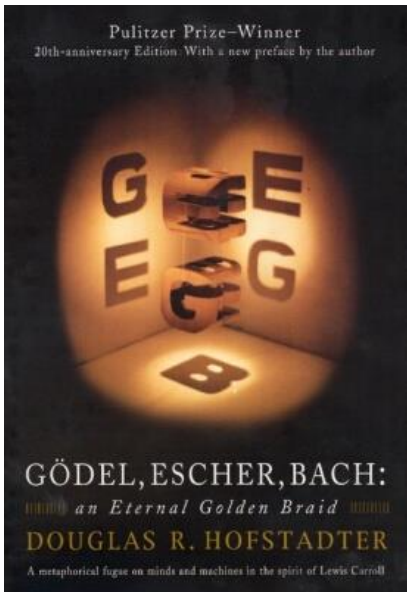
- One of the big questions in Math and CS: **Is $P = NP$?**
 - Or are there problems which cannot be solved in polynomial time.
 - Big practical impact (e.g. in Cryptography).
 - One of the seven **\$1M problems** by the Clay Mathematics Institute of Cambridge, Massachusetts.

Summary (Chomsky Hierarchy)



Bedtime Reading

If you're leaning towards "human = machine"



If you're leaning towards "human \supset machine"