

Entailment and Algorithms

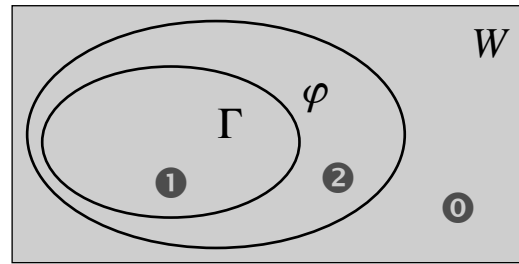
Marco Piastra

Entailment as *Satisfiability*

Transforming problems: entailment as satisfiability

- Step 1: the decision problem “ $\Gamma \models \varphi$?” can be transformed into a *satisfiability* problem

In fact, $\Gamma \models \varphi$ iff $\Gamma \cup \{\neg\varphi\}$ is *not* satisfiable



($w(\Gamma)$ is the set of possible worlds that satisfy Γ)

$$\Gamma \models \varphi \Rightarrow w(\Gamma) \subseteq w(\{\varphi\})$$

$$\mathbf{1} \subseteq \{\mathbf{1}, \mathbf{2}\}$$

$$w(\Gamma \cup \{\neg\varphi\}) = w(\Gamma) \cap w(\{\neg\varphi\})$$

$$w(\{\neg\varphi\}) = \mathbf{0}$$

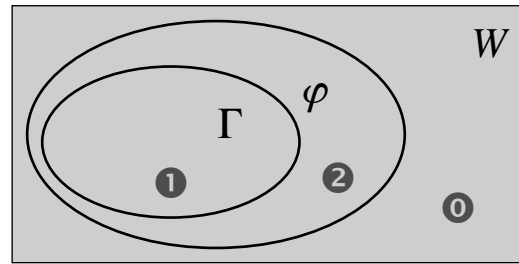
$$w(\Gamma \cup \{\neg\varphi\}) = \emptyset$$

$$\mathbf{1} \cap \mathbf{0} = \emptyset$$

Transforming problems: entailment as satisfiability

- Step 1: the decision problem “ $\Gamma \models \varphi$?” can be transformed into a *satisfiability* problem

In fact, $\Gamma \models \varphi$ iff $\Gamma \cup \{\neg\varphi\}$ is *not* satisfiable



$w(\Gamma)$ is the set of possible worlds that satisfy Γ

$$\Gamma \models \varphi \Rightarrow w(\Gamma) \subseteq w(\{\varphi\})$$

$$\mathbf{1} \subseteq \{\mathbf{1}, \mathbf{2}\}$$

$$w(\{\neg\varphi\}) = \mathbf{0}$$

$$w(\Gamma \cup \{\neg\varphi\}) = w(\Gamma) \cap w(\{\neg\varphi\})$$

$$w(\Gamma \cup \{\neg\varphi\}) = \emptyset$$

$$\mathbf{1} \cap \mathbf{0} = \emptyset$$

- Step 2: the decision problem “is $\Gamma \cup \{\neg\varphi\}$ satisfiable?” can be transformed into a wff *satisfiability* problem

Taking this one step further, we can transform $\Gamma \cup \{\neg\varphi\}$ into *just one formula*:

$$\bigwedge (\Gamma \cup \{\neg\varphi\})$$

← This is the wff obtained by combining all the wffs in $\Gamma \cup \{\neg\varphi\}$ with \bigwedge , it is called the *conjunctive closure* of the set $\Gamma \cup \{\neg\varphi\}$

Computational Complexity Theory (in a Quick Ride)

Turing Machine (A. Turing, 1937)

■ A more precise definition

A non-empty and finite set of *states* S

At each instant the machine is in a state $s \in S$

A non-empty and finite alphabet of *symbols* Q

The alphabet Q includes a *blank*, default symbol b

Each cell in the tape contains a symbol $q \in Q$

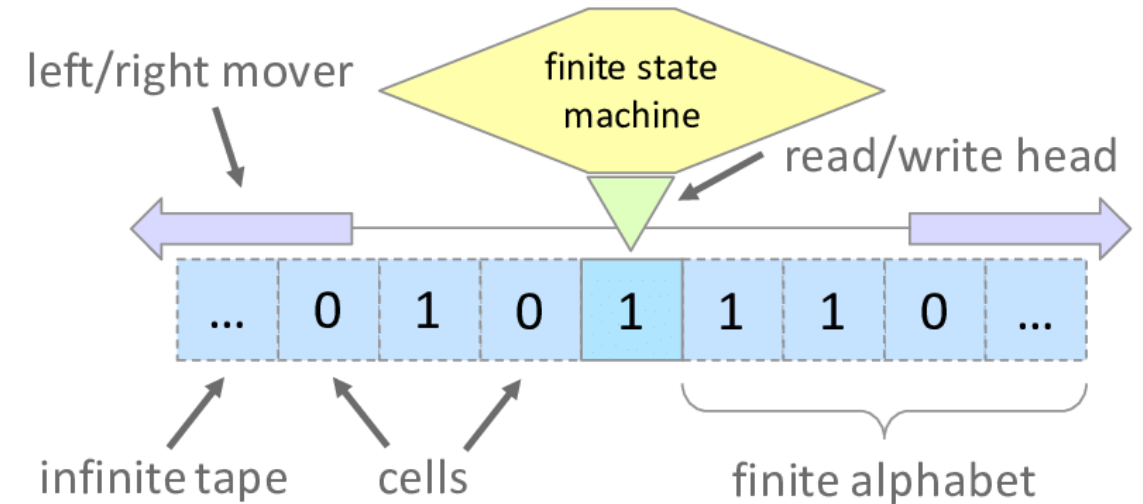
A partial *transition* function

$$\tau : \underbrace{S}_{\text{current state}} \times \underbrace{Q}_{\text{input symbol}} \rightarrow \underbrace{S}_{\text{next state}} \times \underbrace{Q}_{\text{output symbol}} \times \underbrace{\{\text{Left, None, Right}\}}_{\text{head move}}$$

It is partial in the sense it needs not be defined on any input tuple

A subset of *terminal* states $T \subseteq S$

An initial state $s_0 \in S$



Turing Machine (A. Turing, 1937)

- A **busy beaver** example (3 states)

$S = \{A, B, C, \text{HALT}\}$

$s_0 = A$ $T = \{\text{HALT}\}$

$Q = \{0, 1\}$ $b = 0$

$\tau =$

$\langle A, 0 \rangle \rightarrow \langle B, 1, \text{Right} \rangle$

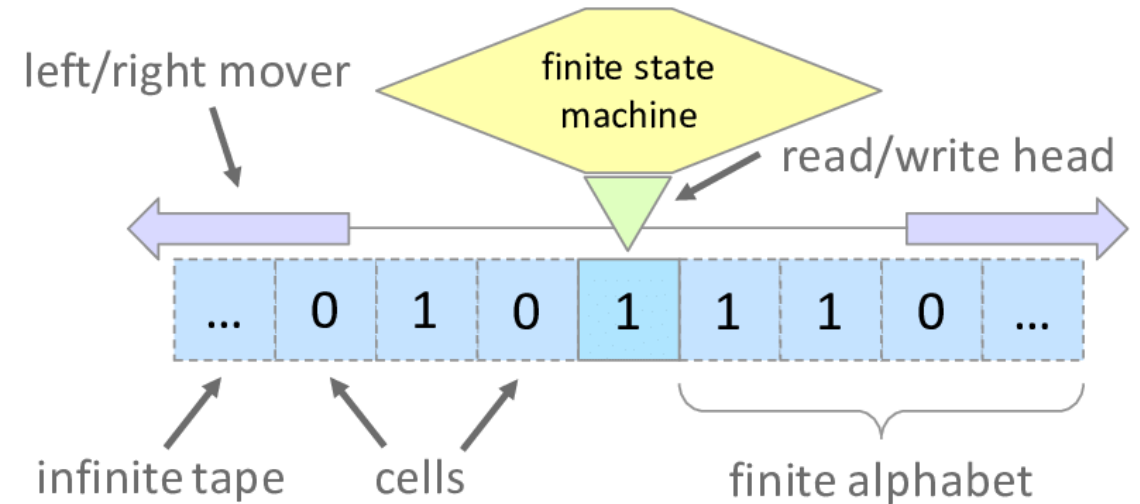
$\langle A, 1 \rangle \rightarrow \langle C, 1, \text{Left} \rangle$

$\langle B, 0 \rangle \rightarrow \langle A, 1, \text{Left} \rangle$

$\langle B, 1 \rangle \rightarrow \langle B, 1, \text{Right} \rangle$

$\langle C, 0 \rangle \rightarrow \langle B, 1, \text{Left} \rangle$

$\langle C, 1 \rangle \rightarrow \langle \text{HALT}, 1, \text{Right} \rangle$



Assume that the tape is infinite and plenty of blank symbols 0

What does this machine do?

Decisions and decidability (automation)

- What is a problem?

A *problem* is an association, i.e. a **relation** between *inputs* and *outputs* (i.e. *solutions*)

$$K = I \times S$$

- *Search* problem

Typically, K associates *one* input to *many* solutions

Optimization problems

A *search problem* plus an *objective* or *cost* function

$$c : S \rightarrow \mathbb{R} \quad (\text{i.e. from } S \text{ to the set of real numbers})$$

In general, the task in a search problem is finding the solution(s) having maximal or minimal cost

- **Decision** problem

The solution space S is $\{0, 1\}$

and K associates each input to a unique solution: $K : I \rightarrow \{0, 1\}$

Example of decision problem: $\Gamma \models \varphi$?

The input space I contains all possible combinations of set Γ of wffs with individual wffs φ

The solution is uniquely defined for any instance of such problems in I

Decisions and decidability (automation)

- **Decidable** problem

A decision problem K for which there exists an algorithm, i.e a *Turing machine*,
(there are other ways of defining an algorithm or an *effective procedure*: they are all equivalent)
that **always terminates** and produces the right answer in **finite time**.

Example of an *undecidable* problem: The *Halting Problem*

Given the formal description of a particular Turing machine and a specific input,
is it possible to tell if whether it will either halt, eventually, or run forever?

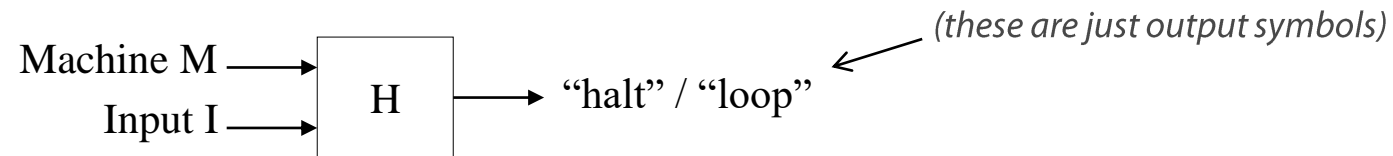
In other words, does it exist a Turing machine that, given in input the description of *another*
Turing machine, will always produce the answer desired?

The answer is **no** (such a Turing machine *cannot* exist)

An aside: The *Halting Problem*

- Intuitive ideas behind the proof (i.e. of the *undecidability* of this problem)

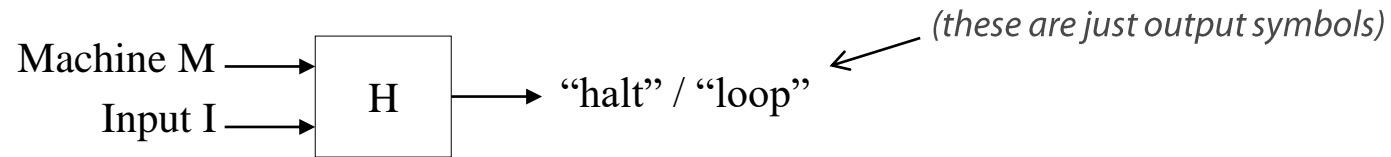
Let's assume there exists a Turing machine H that, given the description of a Turing machine M with input I always terminates producing an output “halt” or “loop” depending on whether M with input I will terminate or not



An aside: The *Halting Problem*

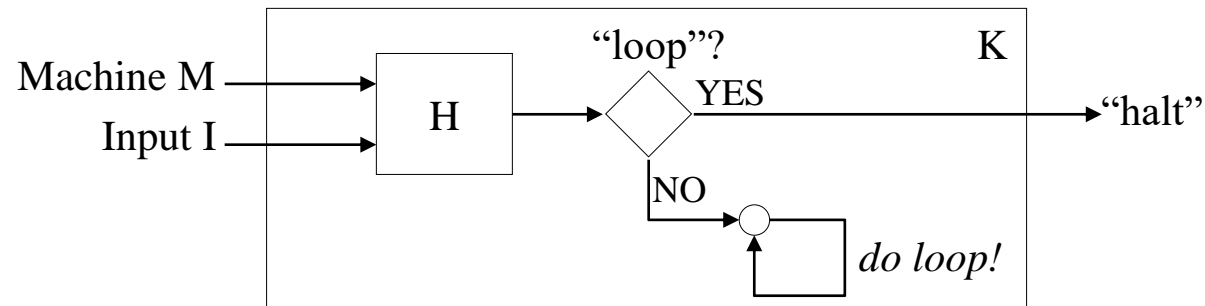
- Intuitive ideas behind the proof (i.e. of the *undecidability* of this problem)

Let's assume there exists a Turing machine H that, given the description of a Turing machine M with input I always terminates producing an output "halt" or "loop" depending on whether M with input I will terminate or not



Assume H existed

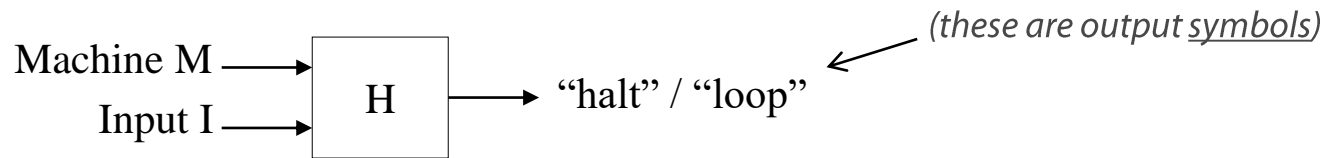
We could build another Turing machine K that enters an infinite loop whenever the output of H is "halt" and that terminates, with output "halt", when H outputs "loop"



An aside: The *Halting Problem*

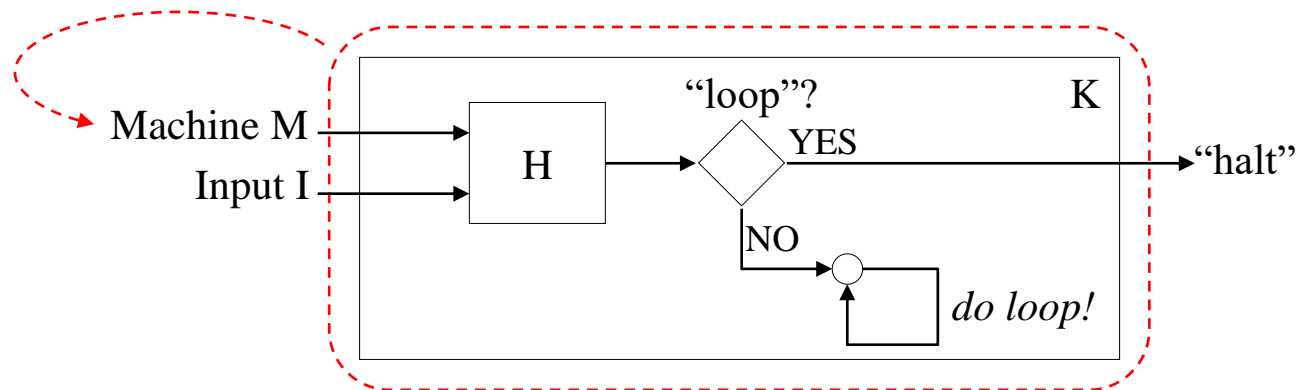
- Intuitive ideas behind the proof (i.e. of the *undecidability* of this problem)

Let's assume there exists a Turing machine H that, given the description of a Turing machine M with input I always terminates producing an output "halt" or "loop" depending on whether M with input I will terminate or not



Assume H existed

We could build another Turing machine K that enters an infinite loop whenever the output of H is "halt" and that terminates, with output "halt", when H outputs "loop"



What will be the output of K when given K *itself* as the input?

K should *diverge* when K *terminates* and vice-versa: i.e. we have an absurdity

Computational complexity

CAUTION: These notions apply to *decidable problems* only

These are related to a (known) Turing machine that gives the answer with respect to the *worst case* scenario (i.e. the less favorable input)

- *Time complexity*

The number of steps that the Turing machine requires for computing the answer, as a function of some numerical dimension of the input (e.g. the number of atoms in a wff)

- *Memory complexity*

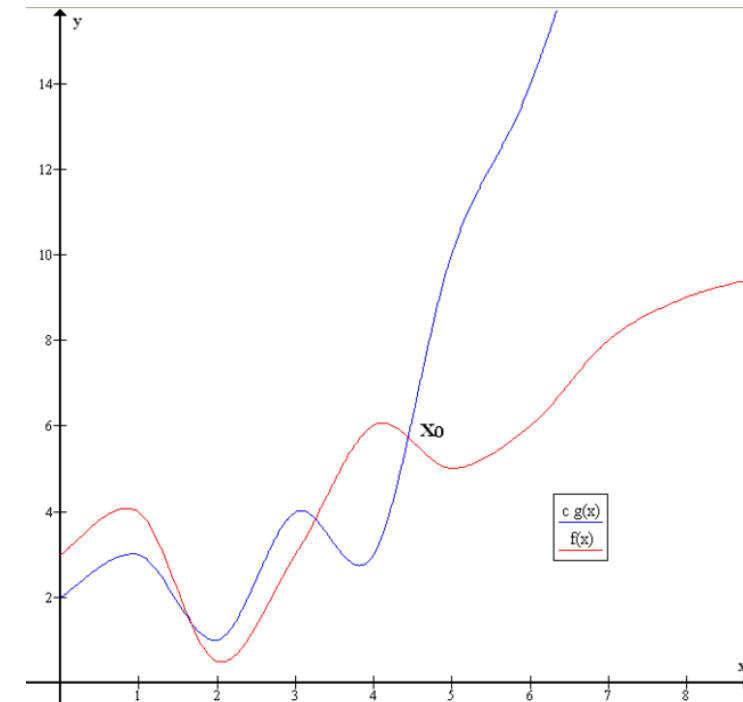
The number of tape cells that the Turing machine requires for computing the answer, as a function of some numerical dimension of the input

- Big-O notation

$$f(x) = O(g(x))$$

means that

$$\exists M > 0, \exists x_0 > 0 \quad \text{such that} \quad |f(x)| \leq M|g(x)|, \quad \forall x > x_0$$



Classes P, NP and NP-complete – The SAT problem

- Class P

The class of problems for which there is a Turing machine that requires $O(P(n))$ time where $P(\cdot)$ is a polynomial of finite degree and n is the dimension of the (*worst-case*) input

- Class NP

The class of all problems:

a) A method for enumerating all possible answers (i.e. *recursive enumerability*)

b) An algorithm in class P that verifies if a possible answer is also a *solution*

It includes all problems in class P (that is, $P \subseteq NP$)

Classes P, NP and NP-complete – The SAT problem

- Class NP-complete

It is a subclass of NP (NP-complete \subseteq NP)

A problem K is NP-complete if every problem in class NP is reducible to K

- Reducibility

For class NP-complete

Consider a problem K for which a decision algorithm $M(K)$ is known

A problem J is reducible to K if there exist a decision algorithm $M(J)$ such that:

- a) algorithm $M(K)$ is called just once, as a “subroutine”, at the end of $M(J)$
- b) apart from $M(K)$, $M(J)$ has polynomial complexity

- The problem SAT

Is NP-complete (*historically, it is the first one to be known*)

Moral: if we had a polynomial decision algorithm for SAT, we would also have that

$$P = NP$$

This fact is not known, it is believed that: $P \neq NP$

(*and a lot will change in the digital world, if this proves to be false*)

Exhaustive (Tree) Search

Satisfiability and decidability (in L_P)

- Is the decision problem “is the wff φ satisfiable?” decidable?

It can be transformed into a *search* problem

i.e. finding a possible world (in the set of all possible worlds) that satisfies φ

In the scientific literature, this problem is called “SAT”

Intuition: we can try every possible value assignment for the atoms in φ

Hint: the problem is NP-complete

Satisfiability and decidability (in L_P)

Example: is this wff satisfiable?

$$\neg(B \vee D \vee \neg(A \wedge C))$$

