# Artificial Intelligence

## An Advanced Course About Foundations




## Entailment and Algorithms

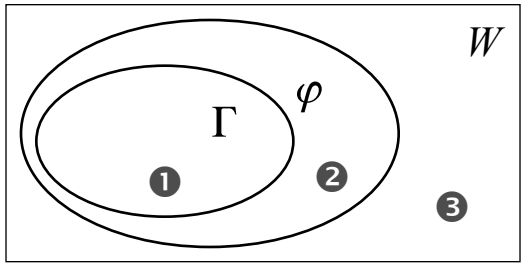
Marco Piastra

# Entailment as *Satisfiability*

# *Transforming problems*: entailment as satisfiability

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

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

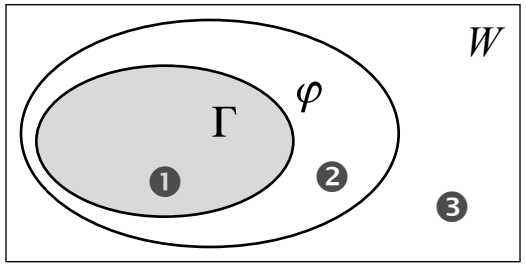



($w(\Gamma)$ is the set of possible worlds that satisfy $\Gamma$)

# *Transforming problems*: Entailment as Satisfiability

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

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



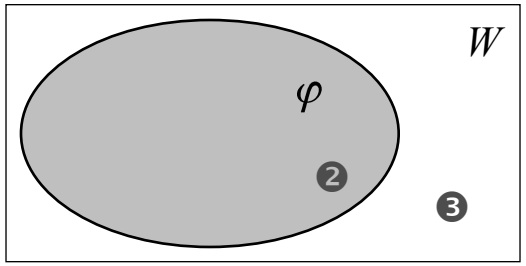

($w(\Gamma)$ is the set of possible worlds that satisfy $\Gamma$)

$w(\Gamma)$ ❶

# *Transforming problems*: Entailment as Satisfiability

- Step 1: the 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 $\Gamma$)

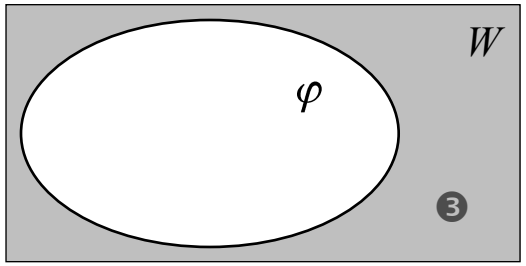
$w(\Gamma)$ ❶

$w(\{\varphi\})$ ❷

# *Transforming problems*: Entailment as Satisfiability

- Step 1: the 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 $\Gamma$)

$w(\Gamma)$ ❶

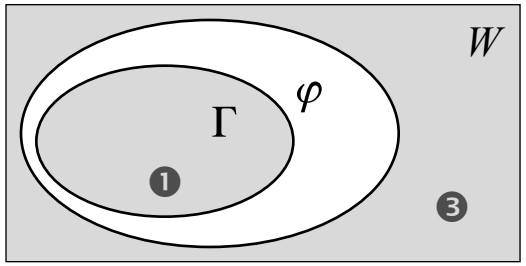
$w(\{\varphi\})$ ❷

$w(\{\neg\varphi\})$ ❸

# *Transforming problems*: Entailment as Satisfiability

- Step 1: the 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 $\Gamma$)

$w(\Gamma)$ ❶

$w(\{\varphi\})$ ❷

$w(\{\neg\varphi\})$ ❸

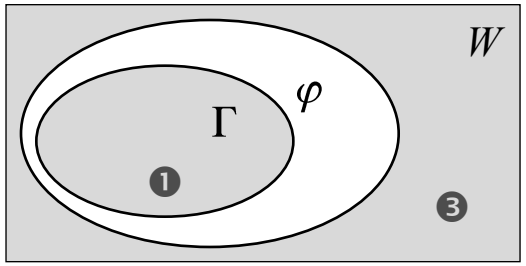
$w(\Gamma \cup \{\neg\varphi\}) = w(\Gamma) \cap w(\{\neg\varphi\})$
$w(\Gamma \cup \{\neg\varphi\}) = \varnothing$

❶ ∩ ❸ = ∅

# *Transforming problems*: Entailment as Satisfiability

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

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



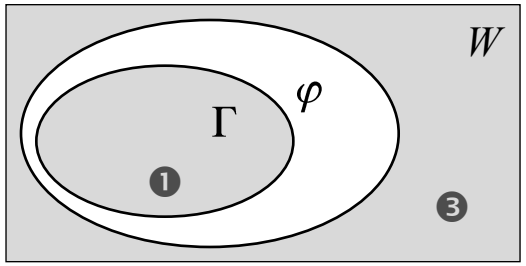

($w(\Gamma)$ is the set of possible worlds that satisfy $\Gamma$)

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

# *Transforming problems*: Entailment as Satisfiability

- Step 1: the 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 $\Gamma$)

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

NOTE: 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 combing all the wffs in $\Gamma \cup \{\neg\varphi\}$ with $\wedge$, it is called the *conjunctive closure* of the set $\Gamma \cup \{\neg\varphi\}$

# *Transforming problems*: Entailment as Satisfiability

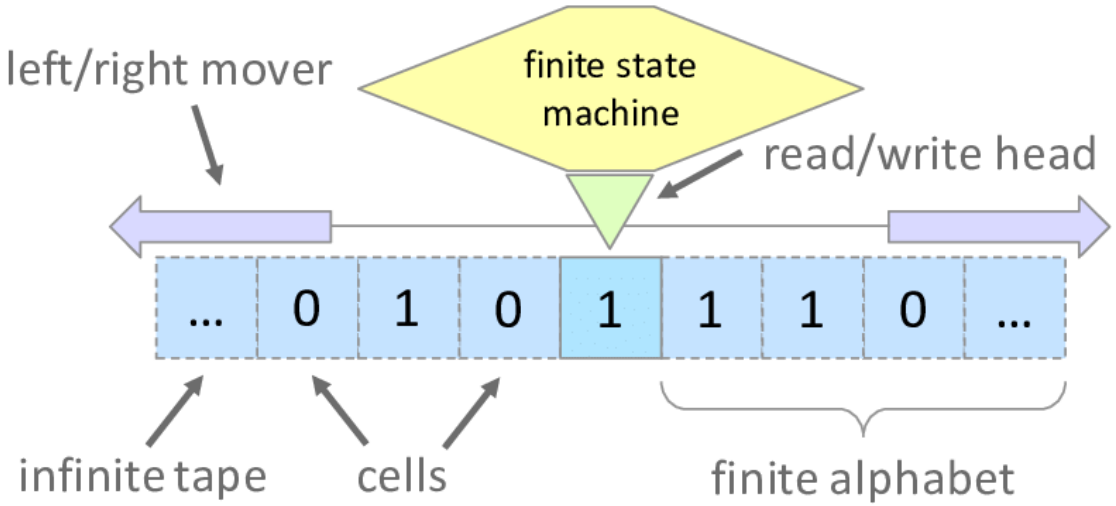
- ***Take-home messages***

  - Each entailment problem $\Gamma \models \varphi$ can be transformed into a *satisfiability* problem
  - There is entailment $\Gamma \models \varphi$, iff $\Gamma \cup \{\neg\varphi\}$ is <u>not</u> satisfiable (=*unsatisfiable*)
  - $\Gamma \cup \{\neg\varphi\}$ is named <u>*refutation set*</u>
  - For computational purposes, the refutation set can be translated into an equivalent $\mathrm{wff}$ via conjunctive closure (=joining all formulae with $\wedge$)
  - The satisfiability problem of a propositional $\mathrm{wff}$ is also called SAT and plays an important role in the field of *computational complexity*

(This last message is new, to be explained)

# "Algorithm"
# (*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 <u>partial</u> *transition* function

$$\tau : S \times Q \rightarrow S \times Q \times \{\text{Left}, \text{None}, \text{Right}\}$$

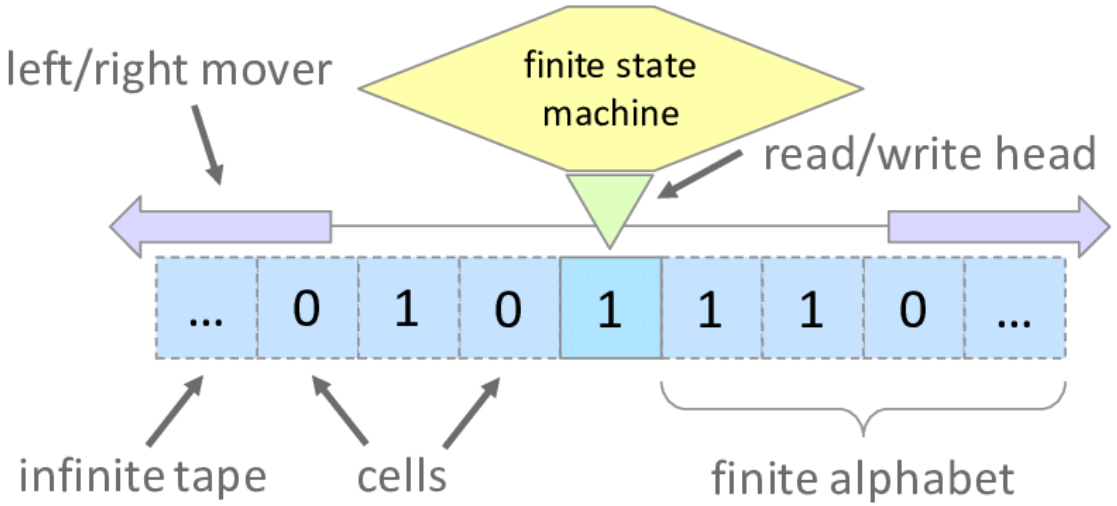
*current state* / *input symbol* / *next state* / *output symbol* / *head move*

*It is <u>partial</u> in the sense it needs not be defined on any input tuple*

  A subset of *terminal* states

  An initial state $s_0 \in S$

# Turing Machine (A. Turing, 1937)




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

$$S = \{A, B, C, \text{HALT}\}$$
$$s_0 = A \qquad T = \{\text{HALT}\}$$
$$Q = \{0, 1\} \qquad b = 0$$
$$\tau =$$
$$< A, 0 > \rightarrow < B, 1, \text{Right} >$$
$$< A, 1 > \rightarrow < C, 1, \text{Left} >$$
$$< B, 0 > \rightarrow < A, 1, \text{Left} >$$
$$< B, 1 > \rightarrow < B, 1, \text{Right} >$$
$$< C, 0 > \rightarrow < B, 1, \text{Left} >$$
$$< C, 1 > \rightarrow < \text{HALT}, 1, \text{Right} >$$

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

*What does this machine do?*

# Decisions and Decidability (automation)

- What is a *problem*?

  A *problem* is a **relation** between *inputs* and *outputs* (= *solutions*)

  $K \subseteq I \times S$

- ***Search problem***

  Typically, $K$ may associate *one* input to *many* solutions

- ***Optimization problem***

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

  $c : S \to \mathbb{R}$ (from $S$ 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 *binary* $\{0, 1\}$ $\quad K : I \to \{0, 1\}$
  and $K$ associates each input to a *unique* solution:

  One remarkable example of decision problem: $\Gamma \models \varphi$ ?

  The input space $\mathrm{I}$ contains all (infinite) combinations of set $\Gamma$ of wffs with one wff $\varphi$
  From theory, the solution (output) is uniquely defined for any instance of such inputs

# 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?

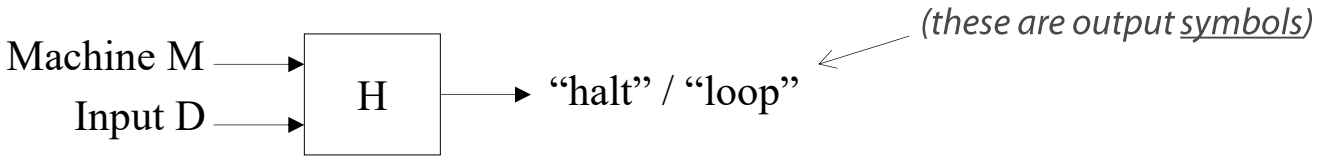
*In other words, does it exist a Turing machine that, given in input the description of another Turing machine, will always produce the answer desired?*

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

# An aside: The *Halting Problem*

- Intuitive idea behind the proof (of the *undecidability* of this problem)

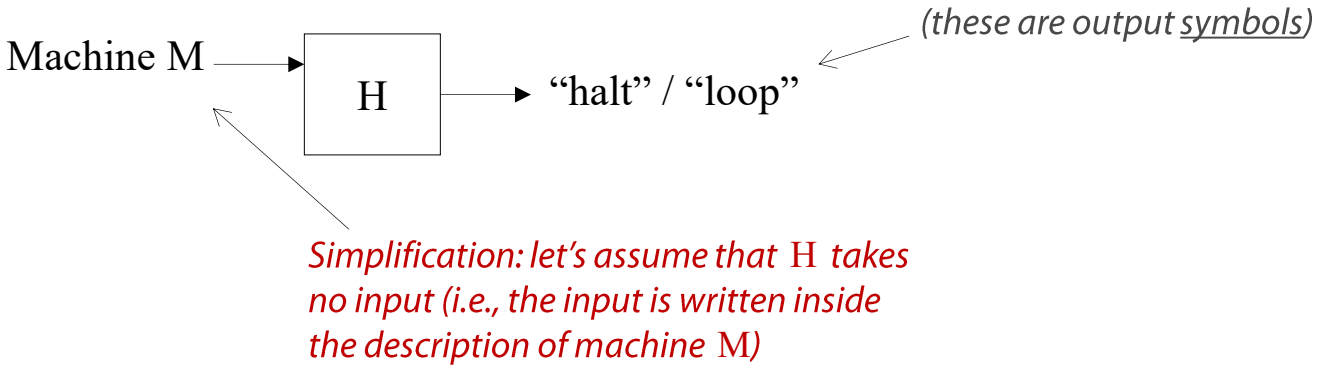
  Let's assume there exists a Turing machine H that, given the description of any Turing machine M with input D always terminates producing an output "halt" or "loop" depending on whether M with input D will terminate or not



# An aside: The *Halting Problem*

- Intuitive idea behind the proof (of the *undecidability* of this problem)

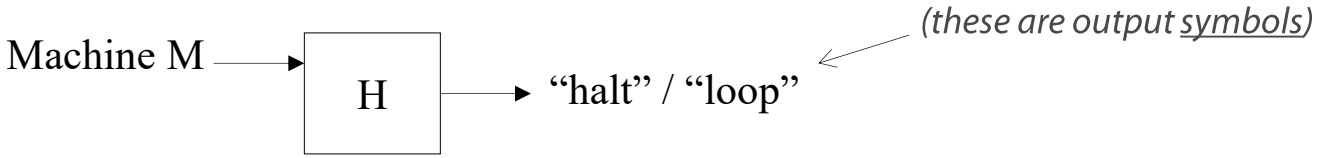
  Let's assume there exists a Turing machine H that, given the description of any Turing machine M with input D always terminates producing an output "halt" or "loop" depending on whether M with input D will terminate or not



# An aside: The *Halting Problem*

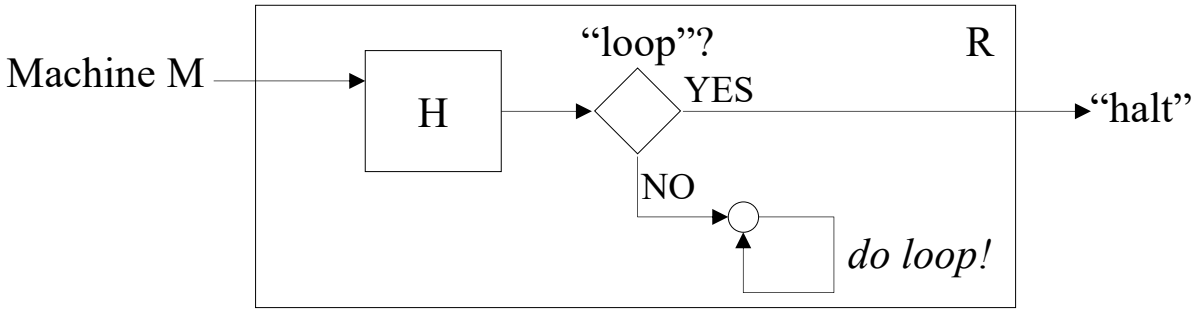
- Intuitive idea behind the proof (of the *undecidability* of this problem)

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




*Assume H existed*

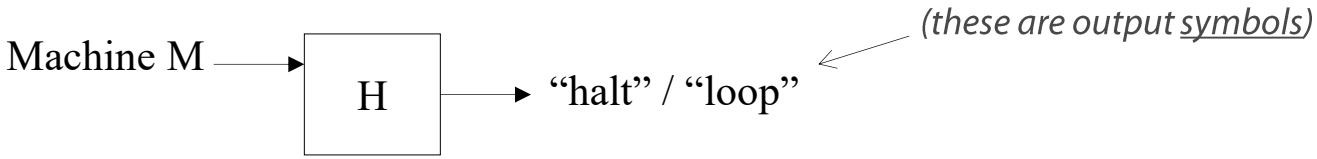
We could build another Turing machine R that enters an infinite loop whenever the output of H is "halt" and that terminates, with output "halt", when H outputs "loop"



# An aside: The *Halting Problem*

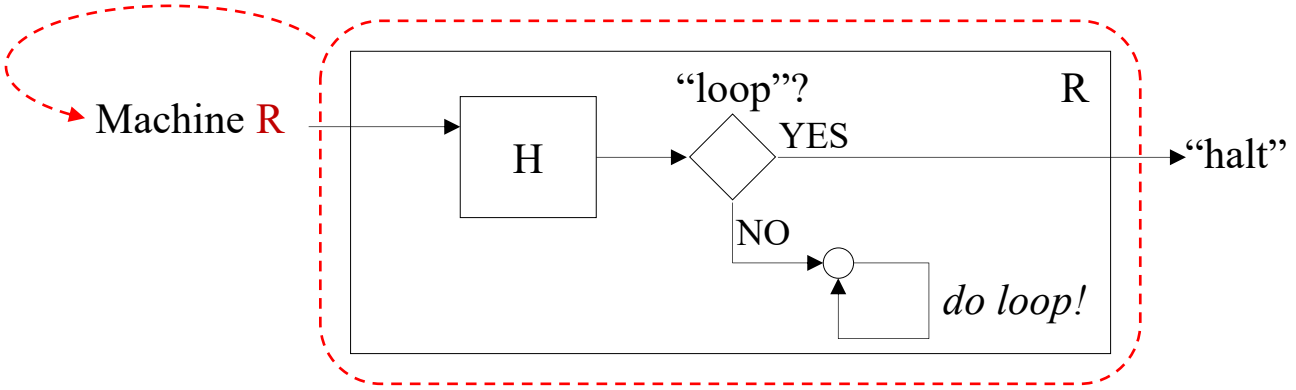
- Intuitive idea behind the proof (of the *undecidability* of this problem)

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




*Assume H existed*

We could build another Turing machine R 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 R when given R *itself* as the input?

R should *diverge* when R *terminates* and vice-versa: we have an absurdity

# Computational Complexity

CAUTION: *These notions apply to decidable problems only*

The benchmark is a (known) Turing machine that computes the correct answer in worst-case scenarios (= the least favorable inputs)

- *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 (example: the number of atoms in a wff)

- *Memory* complexity

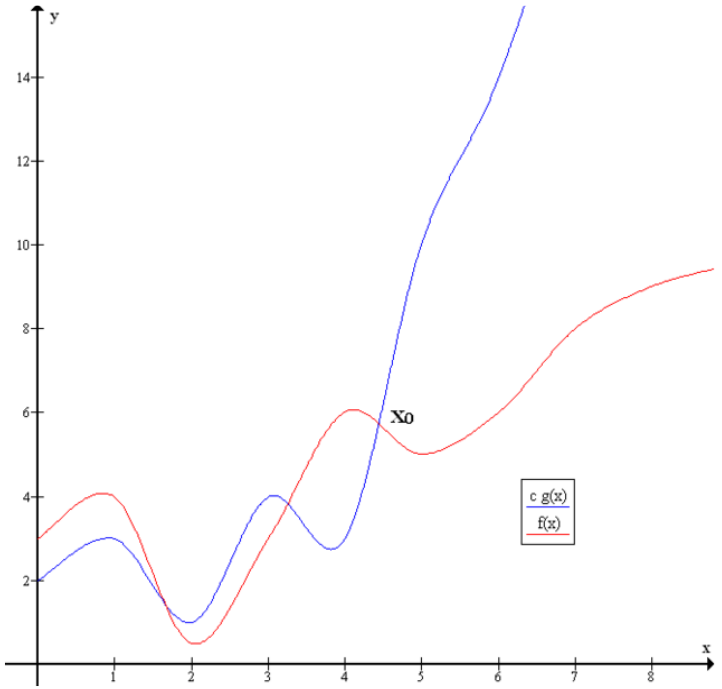
  The number of tape *cells* that the Turing machine requires for computing the answer, as a function of some numerical dimension of the input

- *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 $\mathrm{P}$

  The class of problems for which there is a Turing machine that requires $O(\mathrm{P}(n))$ time

  where $\mathrm{P}(\ )$ is a polynomial of finite degree and $n$ is the dimension of the (*worst-case*) input

- Class $\mathrm{NP}$

  The class of all problems:

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

  b) An algorithm in class $\mathrm{P}$ that *verifies* if a possible answer is also a *solution*

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

# 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 SAT Problem***

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

  If we had a polynomial decision algorithm for SAT, we would also have

  P = NP

  This is not known for sure: it is commonly believed that P ≠ NP

  (*and quite a lot would change in the digital world, should this belief be proven false*)

# Exhaustive (Tree) Search

# Satisfiability and decidability (in $L_P$)

- Decision problem: "is the wff $\varphi$ satisfiable? "?

  It can be transformed into a *search problem*

  i.e., finding a possible world (in the set of *all* possible worlds) that satisfies $\varphi$

  In the scientific literature, this problem is called "SAT"

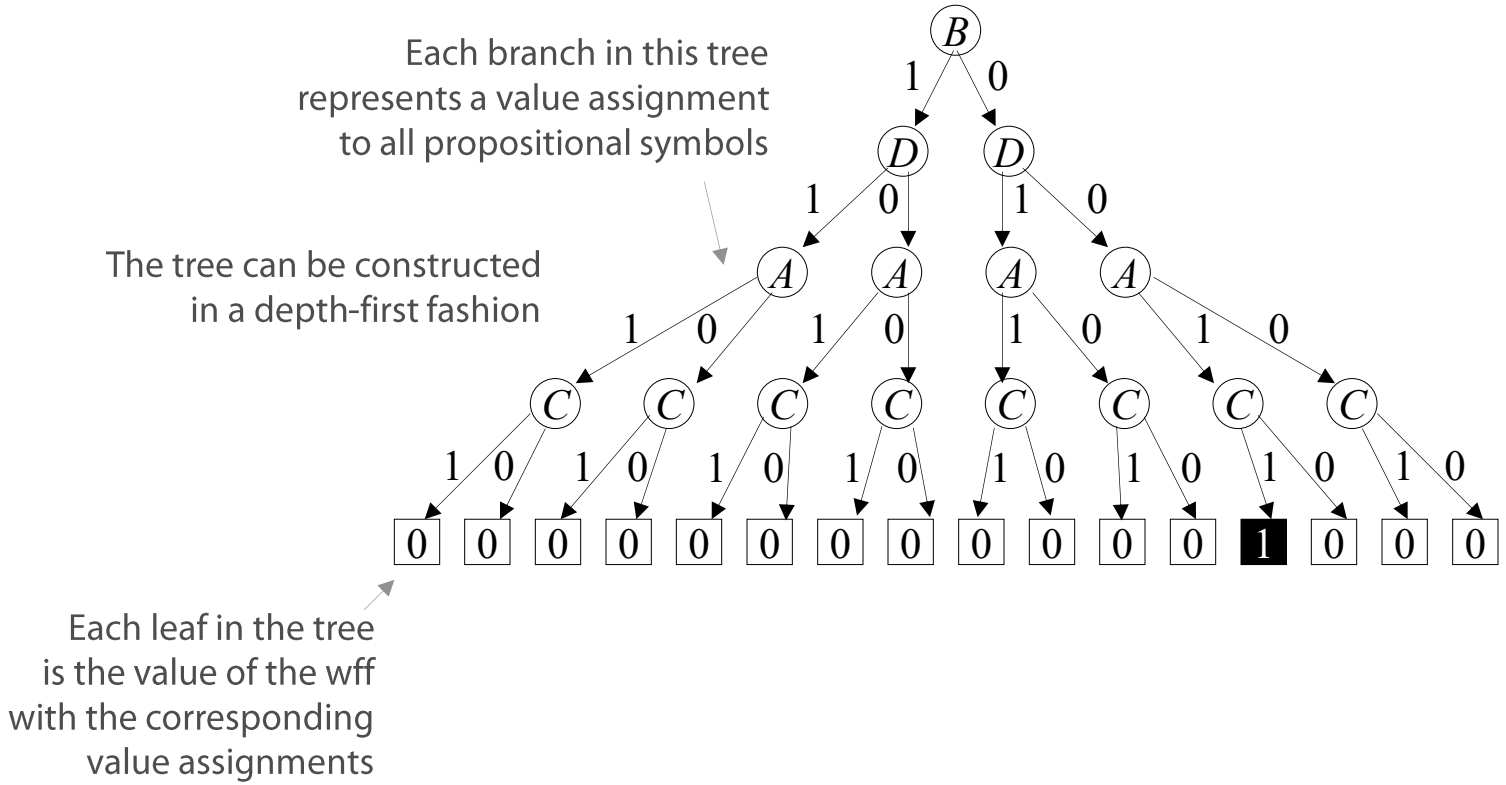
  *Intuition*: we can try every possible value assignment for the atoms in $\varphi$

  *Hint:* the problem is NP-complete

# Satisfiability and decidability (in $L_P$)

Example: is this wff *satisfiable*?

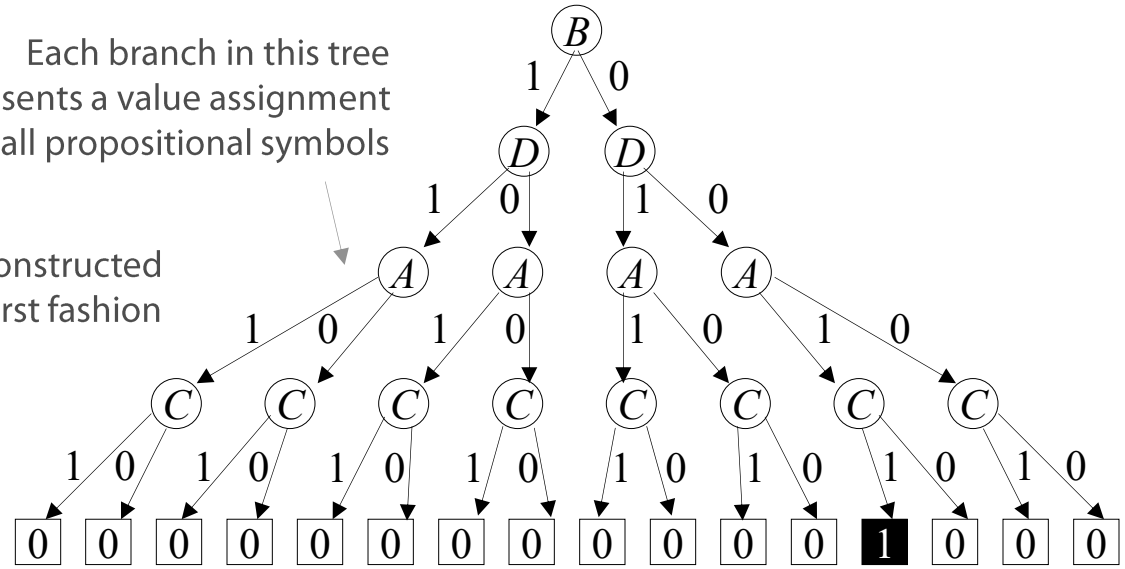
$$\varphi : \neg(B \vee D \vee \neg(A \wedge C))$$



# Satisfiability and decidability (in $L_P$)

Example: is this wff *satisfiable*?

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




Each branch in this tree represents a value assignment to all propositional symbols

The tree can be constructed in a depth-first fashion

Each leaf in the tree is the value of the wff with the corresponding value assignments

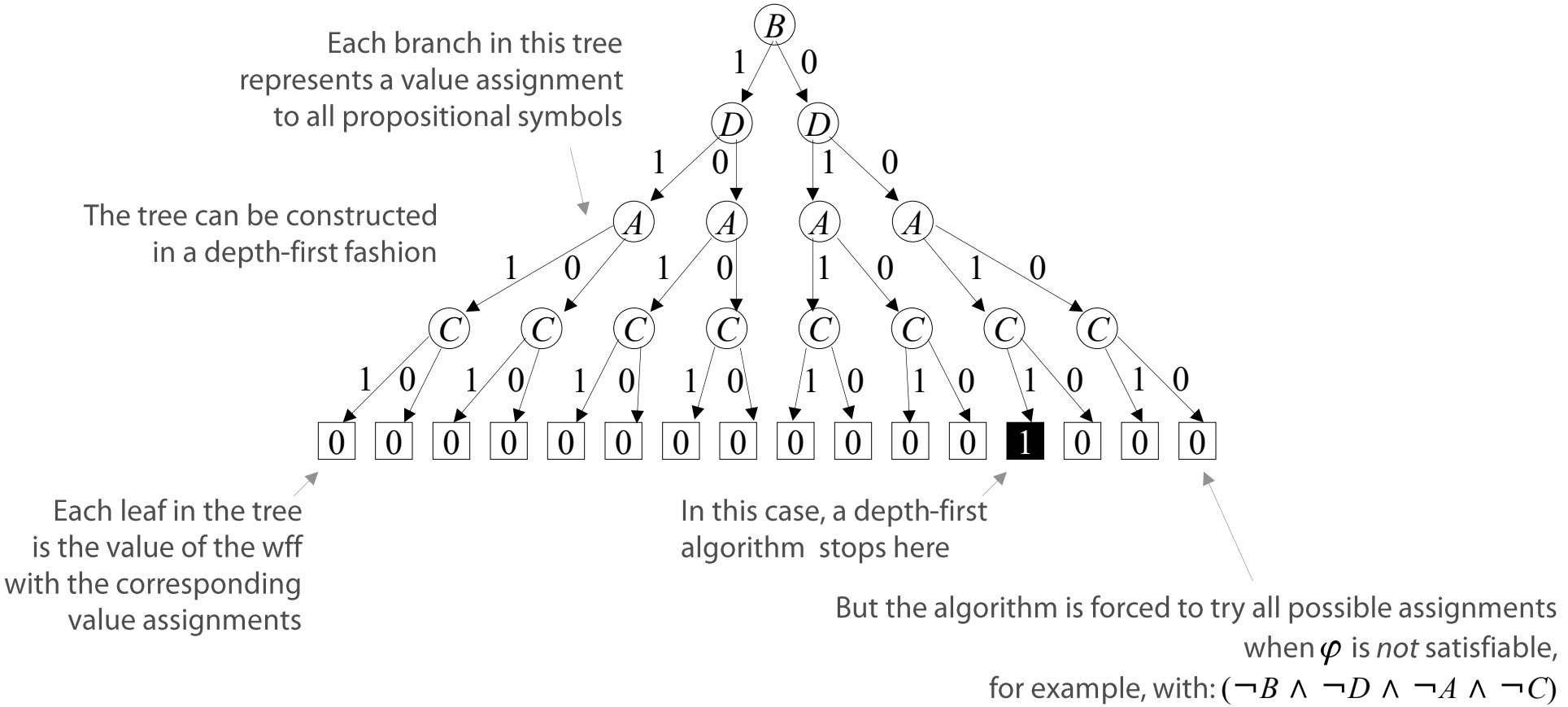
In this case, a depth-first algorithm stops here

But the algorithm is forced to try all possible assignments when $\varphi$ is *not* satisfiable, for example, with: $(\neg B \wedge \neg D \wedge \neg A \wedge \neg C)$

# Satisfiability and decidability (in $L_P$)

Example: is this wff *satisfiable*?

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




This method has $O(2^n)$ time complexity, where $n$ is the number of propositional symbols