NP-complete Problems = "computationally difficult"

⇒ don't expect a poly-time algorithm

Undecidable Problems

= impossible to construct an algorithm that's correct on every input, regardless of running time.

1936 - Alan Turing showed that the Halting Problem is undecidable.

(related to Gödel's incompleteness theorem '31
- roughly, in every mathematical system there is a true but unprovable statement.)

Halting Problem:

input: a program P (written in any language, or can restrict to a certain language) with an input I

output: TRUE if P(I) ever terminates
FALSE if P(I) never terminates
(i.e., has an infinite loop)

Theorem: The Halting Problem is undecidable.

Proof: (by contradiction)

Suppose we had a program that solves the Halting Problem on every input.

Call this program TERMINATES(P, I).

Consider the following evil program:

PARADOX(Z):

(1) If TERMINATES(Z, Z)
    Then GOTO(1)
    Else Return()

What does PARADOX do?

a) if Program Z on input Z terminates eventually
then PARADOX(Z) goes into an infinite loop
& it never terminates.

b) if Z on Z never terminates (i.e., has an infinite loop)
then PARADOX(Z) terminates right away.

Now let Z=PARADOX.

Does PARADOX(PARADOX) terminate?

a) if PARADOX on PARADOX terminates
then PARADOX(PARADOX) never terminates. ⇒⇐

b) if PARADOX on PARADOX never terminates
then PARADOX(PARADOX) terminates ⇒⇐

Either way we get a contradiction so our assumption that TERMINATES() exist must be impossible.

[Cook '71] SAT is NP-complete.

Used Circuit-SAT:

input: DAG representing a circuit where vertices are gates: AND, NOT, OR

AND, OR have in-degree 2

NOT has in-degree 1

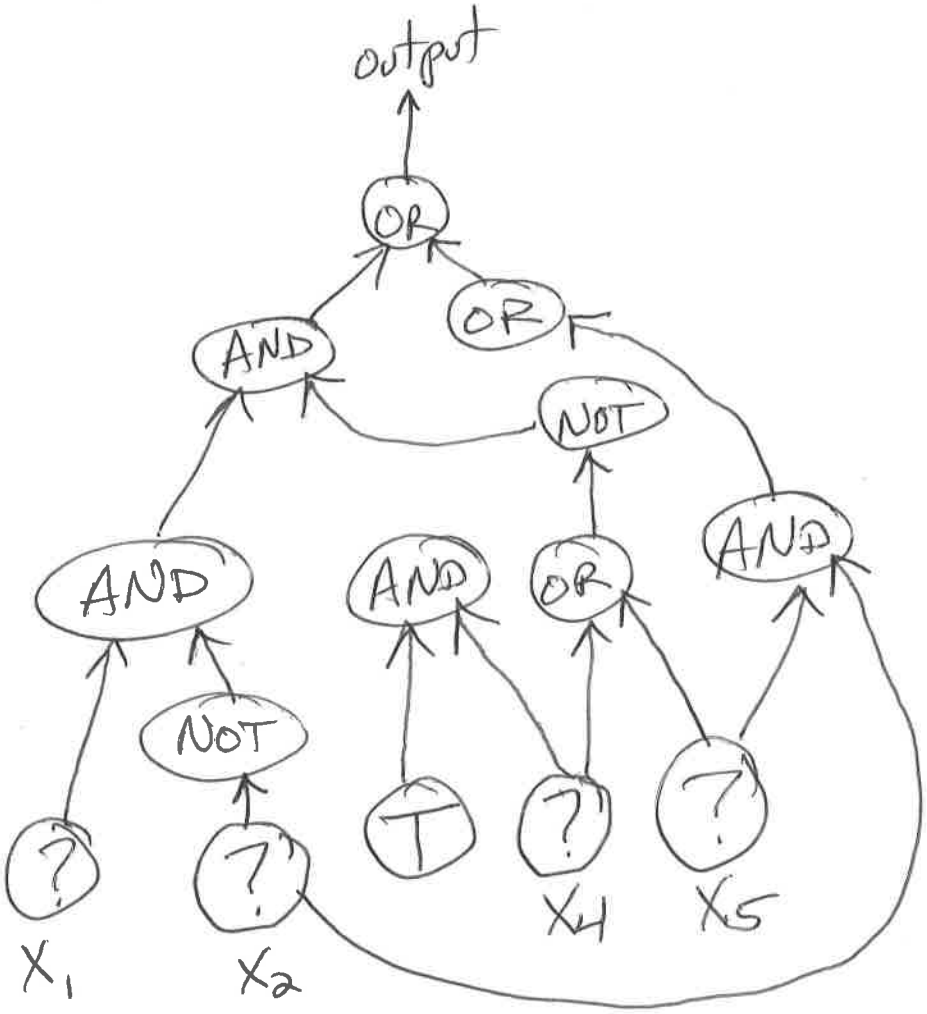
leaves (no in-degree) are labelled T or F or are unlabelled (unknown)

one sink is labelled output.

output: an assignment to unknown inputs so that the circuit evaluates to T

NO if no such assignment.

Example:



Setting $X_1 = T$, $X_2 = F$, $X_4 = F$, $X_5 = F$
then it evaluates to T.

Theorem: Circuit-SAT is NP-complete.

Proof:

a) Circuit-SAT $\in$ NP:

Given a circuit & an assignment, in $O(n+m)$ time we can check that the circuit evaluates to T.

b) for every $A \in NP$, $A \rightarrow$ Circuit-SAT.

A is a search problem so there is an algorithm that "checks" solutions, call this algorithm C. Any algorithm can be represented by a circuit & the unknown gates are used to take the solution S.

Given input I for A, create circuit C_I to check whether a given solution S is a solution to I.

I has a solution iff C_I is satisfiable.

∎

Theorem: SAT is NP-complete

Proof:

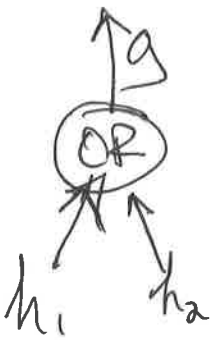
a) SAT $\in$ NP: Given f & an assignment, in $O(n+m)$ time we can verify that the assignment satisfies f.

b) Circuit-SAT $\rightarrow$ SAT:

Given input C with unknown gates $x_1, \ldots, x_n$ as input for Circuit-SAT, our formula f will have variables $x_1, \ldots, x_n$ & we'll create a variable g for each gate in C.
We'll add clauses for these gates as follows:

- If g is an input with known input
  then if input=T add clause (g)
  if input=F add $(\bar{g})$

- If g is an OR of gates h_1 & h_2
  add $(\bar{g} \vee h_1 \vee h_2) \wedge (g \vee \bar{h_1}) \wedge (g \vee \bar{h_2})$



(Note: to satisfy these 3 clauses
if h_1=T or h_2=T set g=T
if $h_1 = h_2$=F set g=F
Thus, $g = h_1 \vee h_2$ so it captures the gate)

for AND gate of h_1 & h_2:

add $(g \vee \overline{h_1} \vee \overline{h_2}) \wedge (\overline{g} \vee h_1) \wedge (\overline{g} \vee h_2)$

(Note to satisfy: if $h_1 = h_2 = T$ then set $g = T$
if $h_1 = F$ then set $g = F$
or
$h_2 = F$

So $g = h_1 \wedge h_2$

for NOT gate of h

$(g \vee h) \wedge (\overline{g} \vee \overline{h})$

if $h = T$ then $g = F$
if $h = F$ then $g = T$

So $g = \overline{h}$