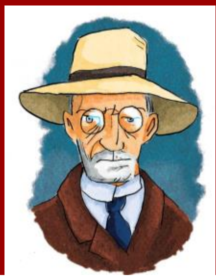
15-251: Great Theoretical Ideas in Computer Science
Lecture 2

Deductive Systems & Propositional Logic



drawing by Alecos Papadatos
color by Annie Di Donna

Remember Tuesday?

Hilbert's 10th problem

Is there a finitary procedure to determine if a given multivariate polynomial with integral coefficients has an integral solution?

Entscheidungsproblem (1928)

Is there a finitary procedure to determine the validity of a given logical expression?

e.g. $\neg\exists x, y, z, n \in \mathbb{N} : (n \geq 3) \wedge (x^n + y^n = z^n)$

(Mechanization of mathematics)

Hilbert's 10th problem

Is there a finitary procedure to determine if a given multivariate polynomia has an integral solution?

Need to talk about Deductive Systems

Entscheidungsproblem (1928)

Is there a finitary procedure to determine the validity of a given logical expression?

e.g. $\neg\exists x, y, z, n \in \mathbb{N} : (n \geq 3$

(Mechanization of mathematics)

Need to talk about Propositional Logic

Deductive Systems

Carrying $0, you walk up to an ATM.

The ATM can dispense:

- any number of $2 bills;
- any number of $5 bills.

Carrying \$0, you walk up to an ATM.

The ATM can dispense:

- any number of \$2 bills;
- any number of \$5 bills.

Which amounts can you leave with?

Solution: Any natural number except 1, 3.

This problem gives a simple example of a **Deductive System**

One **initial object**:
The number 0.

Two **deduction rules**:
i. If x is deducible, then so is x+2.
ii. If x is deducible, then so is x+5.

A **Deductive System** consists of:

- One or more **initial objects**
- One or more **deduction rules**

A **deduction rule** specifies how you may create ("deduce") new objects from ones that you have already created ("deduced").

An example involving parentheses

In this system, objects are **strings** made from the characters (and)

One **initial object**: the string ( )

Two **deduction rules**:
Wrap: from S, may deduce (S)
Concat: from S and T, may deduce S T

One **initial object**: the string ( )
Two **deduction rules**:
Wrap: from S, may deduce (S)
Concat: from S and T, may deduce S T

Problem: Deduce (( ) (( )))

Solution: We may deduce:

( )	[initial object]
(( ))	[*Wrap* applied to ( )]
( ) (( ))	[*Concat* applied to ( ) and (( ))]
(( ) (( )))	[*Wrap* applied to ( ) (( ))]

□

Another example: Defining binary trees

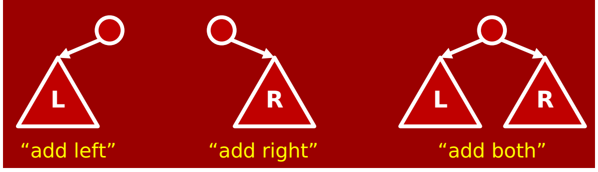
This is a binary tree:

Suppose

and

are binary trees.

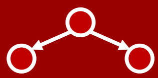
Then each of these is also a binary tree:



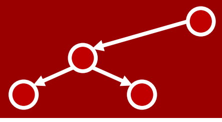
Example binary tree deduction

Initial binary tree:

Apply "add both" deduction, with **L** = **R** =

Hence  is a binary tree.

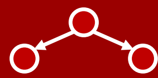
Applying "add left" deduction with above tree…



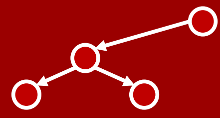
Example binary tree deduction

Applying "add both" to those two deductions:

Apply "add both" deduction, with **L** = **R** =

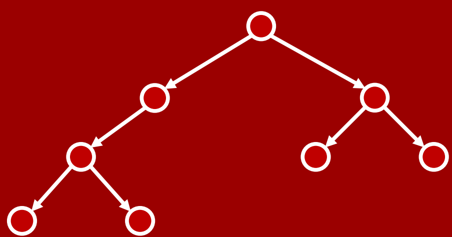
Hence  is a binary tree.

Applying "add left" deduction with above tree…

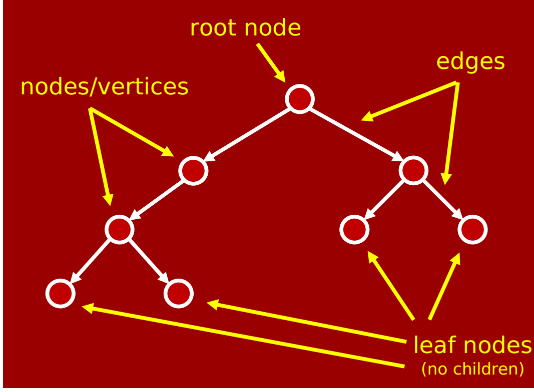


Example binary tree deduction

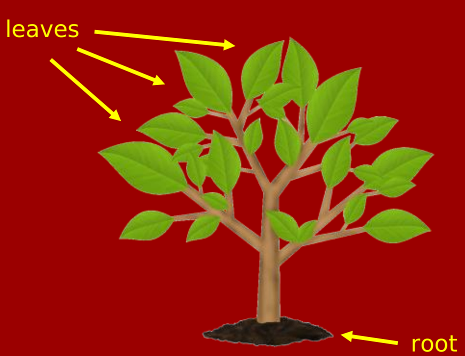
Applying "add both" to those two deductions:



Binary tree terminology



Binary tree terminology



The ATM deductive system

One **initial object**:
The number 0.

Two **deduction rules**:
i. If x is deducible, then so is x+2.
ii. If x is deducible, then so is x+5.

The ATM deductive system

Problem: Show that 4 is deducible

Solution: We may deduce:

0 [initial amount]
2 [+2 rule applied to 0]
4 [+2 rule applied to 2] ∎

The ATM deductive system

Problem: Show that 7 is deducible

Solution: We may deduce:

0 [initial amount]
2 [+2 rule applied to 0]
7 [+5 rule applied to 2] ∎

The ATM deductive system

Problem: Show that 17 is deducible

Solution: We may deduce:

0 [initial amount]
2 [+2 rule applied to 0]
4 [+2 rule applied to 2]
6 [+2 rule applied to 4]
8 [+2 rule applied to 6]
10 [+2 rule applied to 8]
12 [+2 rule applied to 10]
17 [+5 rule applied to 12] ∎

If a specific object *is* deducible, you can always (in principle) show it's deducible by "brute force".

The ATM deductive system

Problem: Show that all nonnegative integers n, $n \neq 1$, $n \neq 3$, are deducible.

There are **infinitely** many objects that we need to show are deducible!

We need **one proof** (written in English) that explains why **all** these deductions are possible.

The ATM deductive system

Problem: Show that all nonnegative integers n, $n \neq 1$, $n \neq 3$, are deducible.

Solution:

Lemma: Suppose n is an **even** nonnegative integer. Then n is deducible.

Proof: Write $n = 2k$, for $k \in \mathbb{N}$. We can deduce n by applying the "+2 rule" k times in succession, starting from 0. ∎

The ATM deductive system

Problem: Show that all nonnegative integers n, $n \neq 1$, $n \neq 3$, are deducible.

Solution:

Lemma: Suppose n is an **even** nonnegative integer. Then n is deducible. ∎

It remains to show that if n is an odd integer and $n \geq 5$, then n is deducible.
Given such an n, let $m = n-5$.
Now m is a nonnegative integer (since $n \geq 5$) and m is even, since it's the difference of two odd #'s.
So by the Lemma, m is deducible.
From this n is deducible, by applying the +5 rule. ∎

The ATM deductive system

Problem: Show that all nonnegative integers n, n ≠ 1, n ≠ 3, are deducible.

Solution: ...... ■

Question: Have we completely characterized the numbers deducible in the ATM deductive system?

No! We have not yet shown that 1 and 3 are **not** deducible!

The ATM deductive system

Problem: Show that 1 and 3 are **not** deducible.

To show that a certain object is **not** deducible, have to write **one** proof showing that **all** possible deductions fail!

Admittedly, it's kind of "obvious" for 1 and 3 in the ATM deductive system, but let's spell it out rigorously.

The ATM deductive system

Problem: Show that 1 and 3 are **not** deducible.

Solution:

We start with 1. Suppose for contradiction that 1 is deducible.
Since 1 is not an initial amount, it would have to be deduced by either the +5 or +2 rule.
But −4 and −1 are not deducible, since all deducible amounts are nonnegative. *[I think this* ***is*** *"obvious".]*

Now we show 3 isn't deducible. Suppose for contradiction it is.
Since 3 is not an initial amount, it would have to be deduced by either the +5 or +2 rule.
It can't be the +5 rule, because −2 is negative.
And it can't be the +2 rule, because we proved 1 is not deducible. ■

Parenthesis deductive system

In this system, objects are **strings** made from the characters **(** and **)**

One **initial object**: the string **( )**

Two **deduction rules**:

Wrap: from **S**, may deduce **(S)**

Concat: from **S** and **T**, may deduce **S T**

Parenthesis deductive system

Suppose I want to show the characterization:

"A string of parenthesis is deducible if and only if it is **balanced**."

What 2 things do I need to prove?

Parenthesis deductive system

"A string of parenthesis is deducible if and only if it is **balanced**."

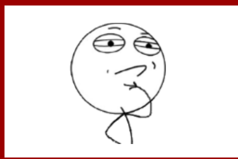
1. Every string of balanced parentheses can be deduced.

   (For this, need to give a method ("algorithm") for generating any given balanced string.)

2. Any string that can be deduced **is** balanced.

   (A pretty straightforward structural induction.)

One final question



"Balanced parentheses" —
what exactly does that mean?

(You will discuss this in recitation tomorrow!)

Propositional Formulas and Circuits

Propositional Logic Refresher

- It's a model for a simple **subset** of mathematical reasoning.
- It's that stuff with formulas like

  $((\neg x \to y) \wedge ((x \vee z) \leftrightarrow y))$

  and truth tables.
- It **doesn't** have "quantifiers": no ∀, ∃. That extension, called "First Order Logic", will be discussed in the next lecture.

Propositional Logic Refresher

First ingredient: **Propositional variables**

Denoted by letters, sometimes with subscripts.
For example, p, w, r, x_1, x_2, x_3, ...

They stand for basic statements that can be either true (**T**) or false (**F**).

E.g.: p stands for "I am playing tennis"
w stands for "I am watching tennis"
r stands for "I am reading about tennis"
x_3 stands for "The 3rd input bit is 1"

Propositional Logic Refresher

Second ingredient: **Connectives**

Not	$\neg$
And	$\wedge$
Or	$\vee$
Implies	$\to$
If And Only If	$\leftrightarrow$

When combined with variables, you get formulas.
For example: $((\neg p \to w) \wedge (\neg w \to r))$

"If I'm not playing tennis then I'm watching tennis, and if I'm not watching tennis then I'm reading about tennis."

Formally defining **formulas**

A well-formed formula over propositional variables $x_1, x_2, \ldots, x_n$ is any string deducible in the following *deductive system*:

Initial formulas: Any variable: $x_1, x_2, \ldots, x_n$

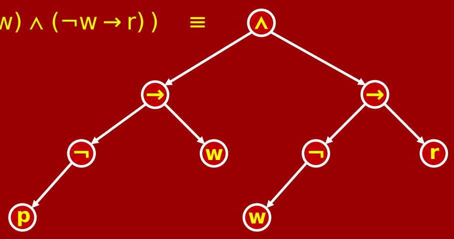
Deduction rules: From A, can obtain $\neg A$
From A, B can obtain $(A \wedge B)$
$(A \vee B)$
$(A \to B)$
$(A \leftrightarrow B)$

E.g.: Show $((\neg p \to w) \wedge (\neg w \to r))$ is a formula.

Equivalently:

A formula is a *binary tree* in which:
2-child nodes are labeled by ∧, ∨, →, or ↔;
1-child nodes are labeled by ¬;
0-child nodes (leaves) are labeled by variables.

$((\neg p \to w) \wedge (\neg w \to r)) \quad \equiv$



Let's talk about **TRUTH**.

"If potassium is observed then carbon and hydrogen are also observed."

$(k \to (c \wedge h))$

Q: Is this statement true?

A: The question does not make sense.

"If potassium is observed then carbon and hydrogen are also observed."

$(k \to (c \wedge h))$

Whether this statement/formula is true/false depends on whether the variables are true/false ("state of the world").

If k is **T**, c is **T**, h is **F**… … the formula is **False**.

If k is **F**, c is **F**, h is **T**… … the formula is **True**.

Truth assignment **V**: setting of **T** or **F** for each variable.

Now given a formula S, we can define its truth value **V**[S] by *structural induction*:

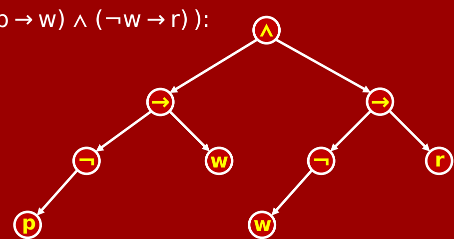
Base case:
If S is a variable x, then **V**[S] is just **V**[x].

Inductive step:
Else S is define by a connective applied to subformulas, and we use the below table:

A	B	¬A	(A∧B)	(A∨B)	(A→B)	(A↔B)
F	F	T	F	F	T	T
F	T	T	F	T	T	F
T	F	F	F	T	F	F
T	T	F	T	T	T	T

In the binary tree perspective:

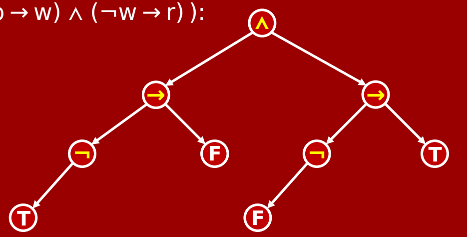
$S = ((\neg p \to w) \wedge (\neg w \to r))$:



Suppose **V** assigns: p to **T**, w to **F**, r to **T**.

In the binary tree perspective:

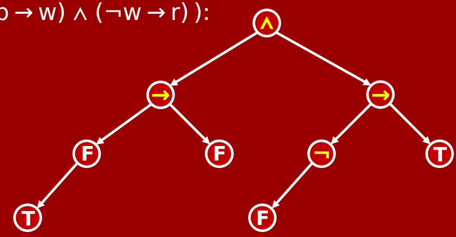
$S = ((\neg p \rightarrow w) \wedge (\neg w \rightarrow r))$:



Suppose **V** assigns: p to **T**, w to **F**, r to **T**.

In the binary tree perspective:

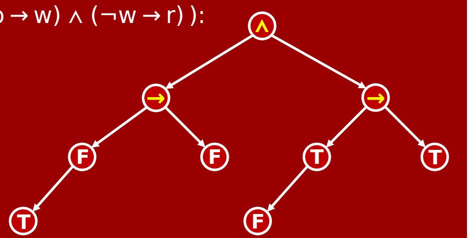
$S = ((\neg p \rightarrow w) \wedge (\neg w \rightarrow r))$:



Suppose **V** assigns: p to **T**, w to **F**, r to **T**.

In the binary tree perspective:

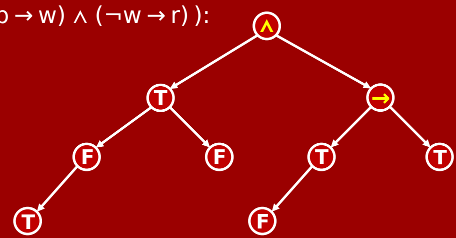
$S = ((\neg p \rightarrow w) \wedge (\neg w \rightarrow r))$:



Suppose **V** assigns: p to **T**, w to **F**, r to **T**.

In the binary tree perspective:

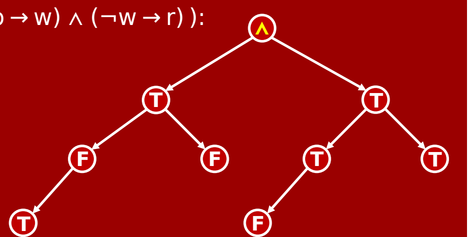
$S = ((\neg p \rightarrow w) \wedge (\neg w \rightarrow r))$:



Suppose **V** assigns: p to **T**, w to **F**, r to **T**.

In the binary tree perspective:

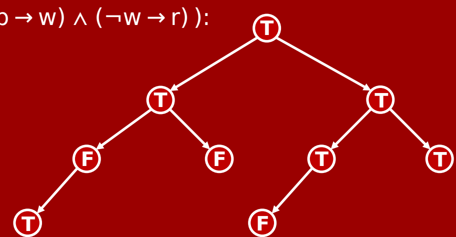
$S = ((\neg p \rightarrow w) \wedge (\neg w \rightarrow r))$:



Suppose **V** assigns: p to **T**, w to **F**, r to **T**.

In the binary tree perspective:

$S = ((\neg p \rightarrow w) \wedge (\neg w \rightarrow r))$:



Suppose **V** assigns: p to **T**, w to **F**, r to **T**.

It follows that **V**[S] = **T**.

Satisfiability

V satisfies S:

V[S] = **T**

S is **satisfiable**:

there exists **V** such that **V**[S] = **T**

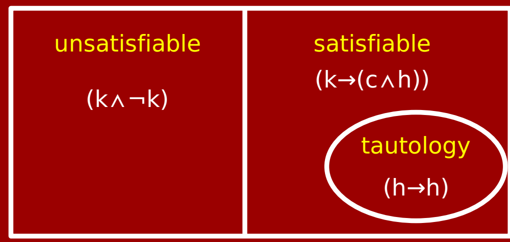
S is **unsatisfiable**:

V[S] = **F** for all **V**

S is a **tautology**:

V[S] = **T** for all **V**

All well-formed formulas



"Potassium is observed and potassium is not observed."

"If potassium is observed then carbon and hydrogen are observed."

"If hydrogen is observed then hydrogen is observed."

Tautology: automatically true, for 'purely logical' reasons

Unsatisfiable: automatically false, for purely logical reasons

Satisfiable (but not a tautology):

truth value depends on the state of the world

S = ((x→(y→z))↔((x∧y)→z))

Truth table

x	y	z	((x→(y→z))↔((x∧y)→z))
F	F	F	
F	F	T	
F	T	F	
F	T	T	
T	F	F	
T	F	T	
T	T	F	
T	T	T	

S = ((x→(y→z))↔((x∧y)→z))

Truth table

x	y	z	((x→(y→z))↔((x∧y)→z))
F	F	F	T
F	F	T	
F	T	F	
F	T	T	
T	F	F	
T	F	T	
T	T	F	
T	T	T	

S is **satisfiable**!

S = ((x→(y→z))↔((x∧y)→z))

Truth table

x	y	z	((x→(y→z))↔((x∧y)→z))
F	F	F	T
F	F	T	T
F	T	F	T
F	T	T	T
T	F	F	T
T	F	T	T
T	T	F	T
T	T	T	T

S is a **tautology**!

Deciding Satisfiability / Tautology

Truth table method:

Pro: Always works

Con: If S has n variables, takes $\approx 2^n$ time

Conjectures: **"P ≠ NP"**

There is **no** polynomial time algorithm that works for every formula.

There is **no** $O(1.999^n)$ time algorithm that works for every formula. **"ETH"**

Another open problem about truth tables: who invented them?

Russell?

Wittgenstein?

Post?

Peirce?

Łukasiewicz?

Jevons?

Ladd–Franklin?

Logical Equivalence

Definition:

Formulas R and S are **equivalent**,

written $R \equiv S$,

if $\mathbf{V}[R] = \mathbf{V}[S]$ for all truth-assignments $\mathbf{V}$.

I.e., their truth tables are exactly the **same**.

Example equivalences

$\neg(x \wedge y) \equiv (\neg x \vee \neg y)$

$\left.\begin{array}{l}\neg(A \wedge B) \equiv (\neg A \vee \neg B) \\ \neg(A \vee B) \equiv (\neg A \wedge \neg B)\end{array}\right\}$ "De Morgan's Laws"

$A \rightarrow B \equiv (\neg A \vee B)$

$A \leftrightarrow B \equiv ((A \rightarrow B) \wedge (B \rightarrow A))$

$\neg\neg A \equiv A$

$(A \vee B) \equiv (B \vee A)$ "commutativity"

$((A \vee B) \vee C) \equiv (A \vee (B \vee C))$ "associativity"

remark: so it's okay to write $(A \vee B \vee C)$

commutativity and associativity of $\wedge$

$A \vee A \equiv A$

etc…

Problem: Show $(((x \rightarrow y) \wedge x) \rightarrow y)$ is a tautology.

Solution 1: Truth-table method

Solution 2: Use equivalences:

$(((x \rightarrow y) \wedge x) \rightarrow y)$

$\equiv \neg((x \rightarrow y) \wedge x) \vee y$ (using $A \rightarrow B \equiv \neg A \vee B$)

$\equiv (\neg(x \rightarrow y) \vee \neg x) \vee y$ (using $\neg(A \wedge B) \equiv \neg A \vee \neg B$)

$\equiv \neg(x \rightarrow y) \vee (\neg x \vee y)$ (using $(A \vee B) \vee C \equiv A \vee (B \vee C)$)

$\equiv \neg(\neg x \vee y) \vee (\neg x \vee y)$ (using $A \rightarrow B \equiv \neg A \vee B$)

$= \neg S \vee S$, where $S = (\neg x \vee y)$.

And a formula of the form $\neg S \vee S$ is always a tautology.

Logical entailment

"Is S a tautology?"

moderately interesting

"Assuming formulas $A_1, \ldots, A_m$ ('axioms') is S a logical consequence ('theorem')?"

more typical kind of thing to be interested in

Logical entailment

Definition:

Formulas $A_1, \ldots, A_m$ **entail** formula S,

written $A_1, \ldots, A_m \vDash S$,

if every truth-assignment ς which makes

$A_1, \ldots, A_m$ equal **T** also makes S equal **T**.

"S is a logical consequence of $A_1, \ldots, A_m$."

Entailment examples

$x, y \vDash (x \wedge y)$

$A, B \vDash (A \wedge B)$

$A \vDash (A \vee B)$ for any B

$A, A \rightarrow B \vDash B$

$A \rightarrow B, B \rightarrow C \vDash A \rightarrow C$

$A \vee x, B \vee \neg x \vDash A \vee B$

etc.

fact: $A_1, \ldots, A_m \vDash S$

iff $(A_1 \wedge \cdots \wedge A_m) \rightarrow S$ is a tautology

From logic to **computation**…

…where we usually write **0** and **1**,
rather than **F** and **T**.

Every formula has a corresponding truth table.

$((x \wedge y) \vee (x \wedge z)) \vee (y \wedge z)$

Every formula has a corresponding truth table.

length-n binary strings

x	y	z	$((x \wedge y) \vee (x \wedge z)) \vee (y \wedge z)$
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Truth table also represents a Boolean function,

$f : \{0,1\}^n \rightarrow \{0,1\}$

A Boolean function $f : \{0,1\}^3 \rightarrow \{0,1\}$ can be specified by a truth table. E.g.:

x	y	z	f(x,y,z)
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Or it can be specified by words. E.g.:
"f(x,y,z) = 1 iff at least two input bits are 1"

Question:

How many Boolean functions (truth tables) are there on n variables?

Answer: 2^{2^n}

We know each propositional formula on n variables "computes" one such function.

Question:

Is every Boolean function (truth table) computed by some propositional formula?

Is every truth table computed by some formula?

x_1	x_2	x_3	x_4	f
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

$x_1 \wedge x_2 \wedge x_3 \wedge x_4$

Is every truth table computed by some formula?

x_1	x_2	x_3	x_4	f
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

$\neg x_1 \wedge \neg x_2 \wedge \neg x_3 \wedge \neg x_4$

Is every truth table computed by some formula?

x_1	x_2	x_3	x_4	f
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

$x_1 \wedge \neg x_2 \wedge x_3 \wedge \neg x_4$

Is every truth table computed by some formula?

x_1	x_2	x_3	x_4	f
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

$\neg x_1 \wedge x_2 \wedge x_3 \wedge x_4$

Is every truth table computed by some formula?

x_1	x_2	x_3	x_4	f
0	0	0	0	
0	0	0	1	
0	0	1	0	
0	0	1	1	
0	1	0	0	
0	1	0	1	
0	1	1	0	
0	1	1	1	
1	0	0	0	
1	0	0	1	
1	0	1	0	
1	0	1	1	
1	1	0	0	
1	1	0	1	
1	1	1	0	
1	1	1	1	

We can similarly do any truth table with exactly one 1.

Is every truth table computed by some formula?

x_1	x_2	x_3	x_4	f
0	0	0	0	**0**
0	0	0	1	**0**
0	0	1	0	**0**
0	0	1	1	**0**
0	1	0	0	**0**
0	1	0	1	**0**
0	1	1	0	**0**
0	1	1	1	**1**
1	0	0	0	**0**
1	0	0	1	**0**
1	0	1	0	**1**
1	0	1	1	**0**
1	1	0	0	**0**
1	1	0	1	**0**
1	1	1	0	**0**
1	1	1	1	**0**

What if there are **two** 1's?

$(\neg x_1 \wedge x_2 \wedge x_3 \wedge x_4)$

$\vee$

$(x_1 \wedge \neg x_2 \wedge x_3 \wedge \neg x_4)$

Is every truth table computed by some formula?

x_1	x_2	x_3	x_4	f
0	0	0	0	**0**
0	0	0	1	**0**
0	0	1	0	**0**
0	0	1	1	**0**
0	1	0	0	**0**
0	1	0	1	**0**
0	1	1	0	**0**
0	1	1	1	**1**
1	0	0	0	**0**
1	0	0	1	**0**
1	0	1	0	**1**
1	0	1	1	**0**
1	1	0	0	**0**
1	1	0	1	**0**
1	1	1	0	**0**
1	1	1	1	**0**

What if there are **three** 1's?

$(\neg x_1 \wedge x_2 \wedge x_3 \wedge x_4)$

$\vee$

$(x_1 \wedge \neg x_2 \wedge x_3 \wedge \neg x_4)$

Is every truth table computed by some formula?

x_1	x_2	x_3	x_4	f
0	0	0	0	**0**
0	0	0	1	**0**
0	0	1	0	**0**
0	0	1	1	**0**
0	1	0	0	**0**
0	1	0	1	**0**
0	1	1	0	**0**
0	1	1	1	**1**
1	0	0	0	**0**
1	0	0	1	**0**
1	0	1	0	**1**
1	0	1	1	**0**
1	1	0	0	**0**
1	1	0	1	**0**
1	1	1	0	**1**
1	1	1	1	**0**

What if there are **three** 1's?

$(\neg x_1 \wedge x_2 \wedge x_3 \wedge x_4)$

$\vee$

$(x_1 \wedge \neg x_2 \wedge x_3 \wedge \neg x_4)$

$\vee$

$(x_1 \wedge x_2 \wedge x_3 \wedge \neg x_4)$

We have just done "proof by example" ☺
for the following result:

Theorem:
Every Boolean function (truth table) over
n variables can be computed by a formula.
(And only using $\neg$, $\wedge$, $\vee$.)

Actually, we missed a case…
…the Boolean function which is always 0.

Well, it's computed by $(x_1 \wedge \neg x_1)$.

Circuits

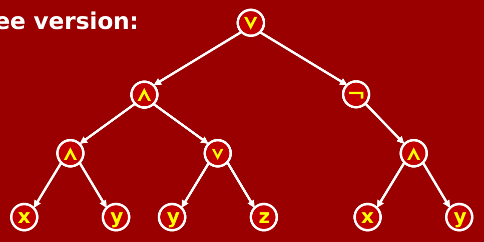
$((x \wedge y) \wedge (y \vee z)) \vee \neg(x \wedge y)$ is a formula.

Deduction: We can deduce it as follows:

x	[variable]
y	[variable]
z	[variable]
$(x \wedge y)$	[$\wedge$ applied to x, y]
$(y \vee z)$	[$\vee$ applied to y, z]
$((x \wedge y) \wedge (y \vee z))$	[$\wedge$ of previous two]
$\neg(x \wedge y)$	[$\neg$ of $(x \wedge y)$]
$((x \wedge y) \wedge (y \vee z)) \vee \neg(x \wedge y)$	[$\vee$ of previous two]

$((x \wedge y) \wedge (y \vee z)) \vee \neg(x \wedge y)$ is a formula.

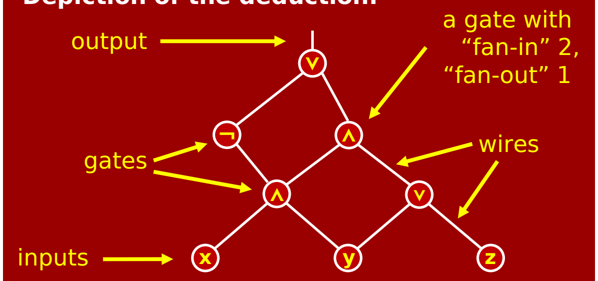
Tree version:



This tree depicts the formula, not the deduction.

$((x \wedge y) \wedge (y \vee z)) \vee \neg(x \wedge y)$ is a formula.

Depiction of the deduction:



Such a picture is called a Boolean **circuit**.

What is the difference between circuits and formulas?

In circuits, nodes (gates) may have fan-out > 1.
(In particular, they are "dags", not trees.)
Formulas are trees: all nodes have fan-out 1.

Circuits can **reuse** already-computed pieces.
Formulas cannot; everything must be "rebuilt".
So circuits can be "more efficient".

Deduction viewpoint: The circuit is the deduction.
The formula is the last line.

We'll end with a mystery from the field of Theoretical Computer Science.

Circuits are a kind of "programming language".
How efficient can they be?

Consider all truth tables with 42 variables.

It's not hard to show that there exists such a truth table (in fact many) such that the smallest circuit computing it requires
at least 100 billion gates.

But no one explicitly knows such a truth table.

The best explicit example we know is a truth table that requires at least **123** gates.

Study Guide

Deductive systems:
- definitions
- characterizations
- binary tree definitions

Propositional logic:
- formulas
- truth assignments
- valid/satisfiable
- truth-table method
- equivalences
- all functions computable

Circuits:
- definitions