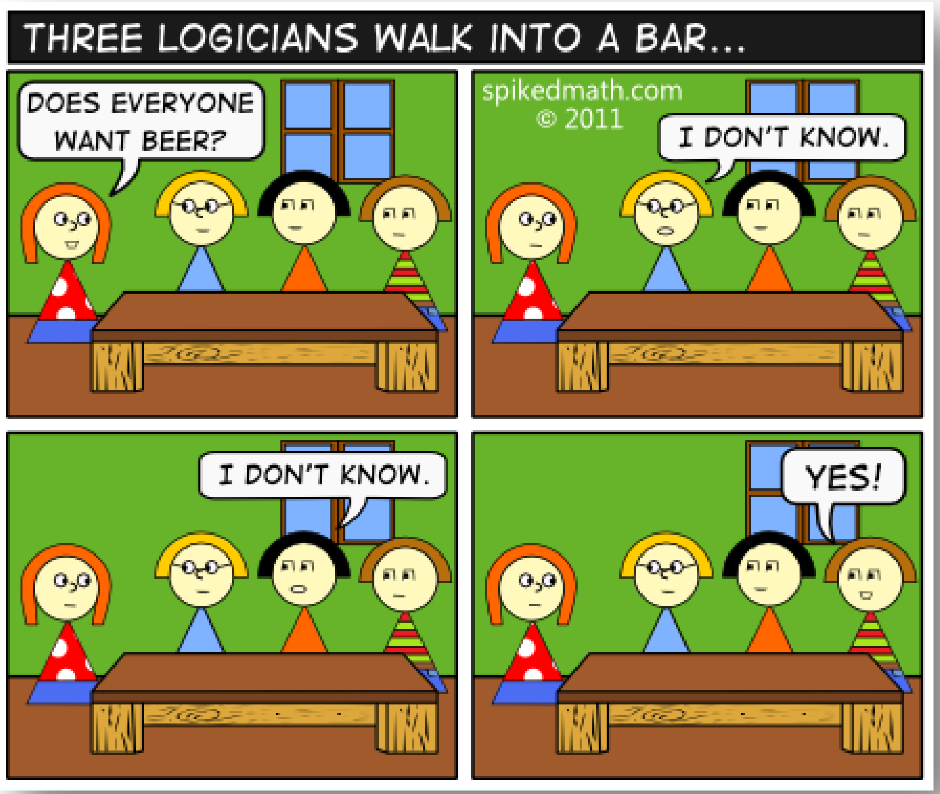
cse 311: foundations of computing

Spring 2015

Lecture 3: Logic and Boolean algebra



gradescope

Homework #1 is up (and has been since Friday).

It is due Friday, October 9th at 11:59pm.

You should have received

(i) An invitation from Gradescope

[if not, email cse311-staff ASAP]

(ii) An email from me about (i)

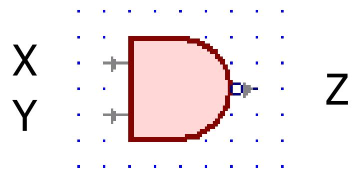
[if not, go to the course web page and sign up for the class email list]

Note: Homework and extra credit are separate assignments.

more gates

NAND

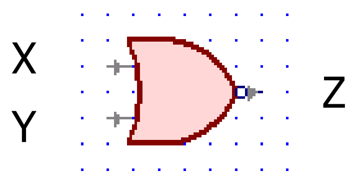
$\neg(X \wedge Y)$



X	Y	Z
0	0	1
0	1	1
1	0	1
1	1	0

NOR

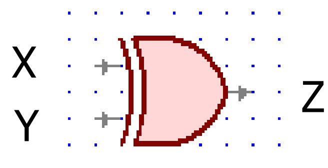
$\neg(X \vee Y)$



X	Y	Z
0	0	1
0	1	0
1	0	0
1	1	0

XOR

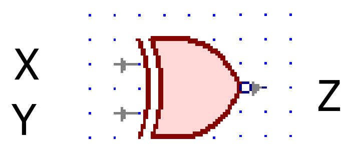
$X \oplus Y$



X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	0

XNOR

$X \leftrightarrow Y, X = Y$



X	Y	Z
0	0	1
0	1	0
1	0	0
1	1	1

Slide 3

1 can we omit this slide? we mentioned it on the hw and this feels like info overload?
Adam Blank, 9/27/2014

Terminology: A compound proposition is a…

- *Tautology* if it is always true
- *Contradiction* if it is always false
- *Contingency* if it can be either true or false

$p \vee \neg p$ Tautology!

$p \oplus p$ Contradiction!

$(p \rightarrow q) \wedge p$ Contingency!

$(p \wedge q) \vee (p \wedge \neg q) \vee (\neg p \wedge q) \vee (\neg p \wedge \neg q)$ Tautology!

A and B are *logically equivalent* if and only if

$A \leftrightarrow B$ is a tautology

i.e. A and B have the same truth table

The notation $A \equiv B$ denotes A and B are logically equivalent.

Example: $p \equiv \neg\neg p$

p	$\neg p$	$\neg\neg p$	$p \leftrightarrow \neg\neg p$
T	F	T	T
F	T	F	T

review: de Morgan's laws

$$\neg (p \lor q) \equiv \neg p \land \neg q$$
$$\neg (p \land q) \equiv \neg p \lor \neg q$$

```
if !(front != null && value > front.data)
    front = new ListNode(value, front);
else {
    ListNode current = front;
    while !(current.next == null || current.next.data >= value)
        current = current.next;
    current.next = new ListNode(value, current.next);
}
```

This code inserts *value* into a sorted linked list.
The first if runs when: front is null or value is smaller than the first item.
The while loop stops when: we've reached the end of the list or the next value is bigger.

review: law of implication

$$(p \to q) \equiv (\neg p \lor q)$$

$$\neg(p \to q) \equiv \neg(\neg p \lor q) \equiv \neg\neg p \land \neg q \equiv p \land \neg q$$

p	q	$p \to q$	$\neg p$	$\neg p \lor q$	$(p \to q) \leftrightarrow (\neg p \lor q)$
T	T	T	F	T	T
T	F	F	F	F	T
F	T	T	T	T	T
F	F	T	T	T	T

computing equivalence

Describe an algorithm for computing if two logical expressions/circuits are equivalent.

What is the run time of the algorithm?

Compute the entire truth table for both of them!

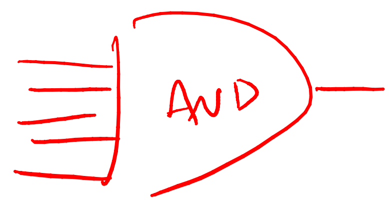
There are 2^n entries in the column for n variables.

$$(p_1 \vee p_2 \wedge p_3) \vee \neg(p_3 \wedge p_4 \vee \neg p_5)$$

some familiar properties of arithmetic

- $x + y = y + x$ (commutativity)
  - $p \vee q \equiv q \vee p$
  - $p \wedge q \equiv q \wedge p$

- $x \cdot (y + z) = x \cdot y + x \cdot z$ (distributivity)
  - $p \wedge (q \vee r) \equiv (p \wedge q) \vee (p \wedge r)$
  - $p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)$



- $(x + y) + z = x + (y + z)$ (associativity)
  - $(p \vee q) \vee r \equiv p \vee (q \vee r)$
  - $(p \wedge q) \wedge r \equiv p \wedge (q \wedge r)$

$$(p_1 \wedge p_2) \wedge (p_3 \wedge p_4) \equiv p_1 \wedge (p_2 \wedge (p_3 \wedge p_4))$$

properties of logical connectives

- Identity
  - $p \wedge \mathrm{T} \equiv p$
  - $p \vee \mathrm{F} \equiv p$
- Domination
  - $p \vee \mathrm{T} \equiv \mathrm{T}$
  - $p \wedge \mathrm{F} \equiv \mathrm{F}$
- Idempotent
  - $p \vee p \equiv p$
  - $p \wedge p \equiv p$
- Commutative
  - $p \vee q \equiv q \vee p$
  - $p \wedge q \equiv q \wedge p$

You will always get this list.

- Associative

  $(p \vee q) \vee r \equiv p \vee (q \vee r)$

  $(p \wedge q) \wedge r \equiv p \wedge (q \wedge r)$
- Distributive

  $p \wedge (q \vee r) \equiv (p \wedge q) \vee (p \wedge r)$

  $p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)$
- Absorption

  $p \vee (p \wedge q) \equiv p$

  $p \wedge (p \vee q) \equiv p$
- Negation

  $p \vee \neg p \equiv \mathrm{T}$

  $p \wedge \neg p \equiv \mathrm{F}$

understanding connectives

- Reflect basic rules of reasoning and logic
- Allow manipulation of logical formulas
  - Simplification
  - Testing for equivalence
- Applications
  - Query optimization
  - Search optimization and caching
  - Artificial intelligence / machine learning
  - Program verification

$$\frac{p \qquad p \rightarrow q}{\therefore\ q}$$

$$(p \wedge (p \rightarrow q)) \rightarrow q$$

equivalences related to implication

$$p \to q \equiv \neg p \vee q$$

$$p \to q \equiv \neg q \to \neg p$$

$$p \leftrightarrow q \equiv (p \to q) \wedge (q \to p)$$

$$p \leftrightarrow q \equiv \neg p \leftrightarrow \neg q$$

$$p \to q \equiv \neg p \vee q$$

$$\neg q \to \neg p \equiv \neg\neg q \vee \neg p$$

$$\equiv q \vee \neg p \equiv \neg p \vee q$$

$$\equiv p \to q$$

logical proofs

To show P is equivalent to Q

– Apply a series of logical equivalences to sub-expressions to convert P to Q

To show P is a tautology

– Apply a series of logical equivalences to sub-expressions to convert P to **T**

$P \equiv T$

$a \vee b \wedge c \equiv a \vee (b \wedge c)$

prove this is a tautology

$$(p \wedge q) \rightarrow (p \vee q)$$

$\equiv \neg(p \wedge q) \vee (p \vee q)$ law of implication

$\equiv (\neg p \vee \neg q) \vee (p \vee q)$ De Morgan

$\equiv (\neg p \vee \neg q) \vee (q \vee p)$ comm.

$\equiv \neg p \vee (\neg q \vee (q \vee p))$ assoc.

$\equiv \neg p \vee ((\neg q \vee q) \vee p)$ assoc.

$\equiv \neg p \vee (T \vee p)$ law of neg.

$\equiv \neg p \vee T$ law of domination

$\equiv T$ " "

prove this is a tautology

$$(p \wedge (p \to q)) \to q$$

$\equiv \neg(p \wedge (p \to q)) \vee q$

$\equiv \neg(p \wedge (\neg p \vee q)) \vee q$

$\equiv (\neg p \vee \neg(\neg p \vee q)) \vee q$

$\equiv (\neg p \vee (\neg\neg p \wedge \neg q)) \vee q$

$\equiv (\neg p \vee (p \wedge \neg q)) \vee q$

$\equiv ((\neg p \vee p) \wedge (\neg p \vee \neg q)) \vee q$

$\equiv (T \wedge (\neg p \vee \neg q)) \vee q$

$\equiv (\neg p \vee \neg q) \vee q$

$\equiv \neg p \vee (\neg q \vee q)$

$\equiv \neg p \vee T$

$\equiv T$.

✓

$\neg(a \wedge b) \equiv (\neg a \vee \neg b)$

prove these are equivalent

$(p \to q) \to r$

$\equiv \neg(p \to q) \vee r$

$\equiv \neg(\neg p \vee q) \vee r$

$\equiv (p \wedge \neg q) \vee r$

$\equiv (p \vee r) \wedge (\neg q \vee r)$

$p \to (q \to r)$

$\equiv \neg p \vee (q \to r)$

$\equiv \neg p \vee \neg q \vee r$

$p = F$

$q = F$

$r = F$

$(F \to F) \to F$

$\equiv T \to F \equiv F$

T

not equiv!

prove these are **not** equivalent

$(p \to q) \to r$

$p \to (q \to r)$

Boolean logic

Combinational Logic

– output = F(input)

Sequential Logic

– $output_t = F(output_{t-1}, input_t)$
  - output dependent on history
  - concept of a time step (clock, t)

Boolean Algebra consisting of…

– a set of elements B = {0, 1}
– binary operations { + , • } (OR, AND)
– and a unary operation { ' } (NOT)

George "homeopathy" Boole

Sessions of class:

We would like to compute the number of lectures or quiz sections remaining *at the start* of a given day of the week.

- Inputs: Day of the Week, Lecture/Section flag
- Output: Number of sessions left

Examples: Input: (Wednesday, Lecture) Output: 2

Input: (Monday, Section) Output: 1

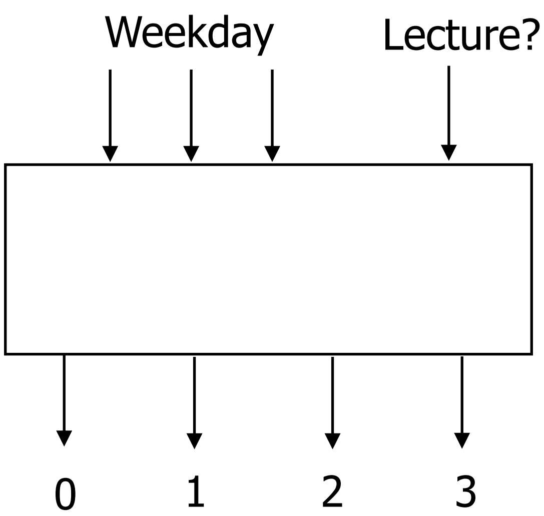
implementation in software

```
public int classesLeft (weekday, lecture_flag) {
    switch (day) {
        case SUNDAY:
        case MONDAY:
            return lecture_flag ? 3 : 1;
        case TUESDAY:
        case WEDNESDAY:
            return lecture_flag ? 2 : 1;
        case THURSDAY:
            return lecture_flag ? 1 : 1;
        case FRIDAY:
            return lecture_flag ? 1 : 0;
        case SATURDAY:
            return lecture_flag ? 0 : 0;
   }
}
```

implementation with combinational logic

Encoding:

- How many bits for each input/output?
- Binary number for weekday
- One bit for each possible output



defining our inputs

```
public int classesLeft (weekday, lecture_flag) {
    switch (day) {
        case SUNDAY:
        case MONDAY:
            return lecture_flag ? 3 : 1;
        case TUESDAY:
        case WEDNESDAY:
             return lecture_flag ? 2 : 1;
        case THURSDAY:
             return lecture_flag ? 1 : 1;
        case FRIDAY:
             return lecture_flag ? 1 : 0;
        case SATURDAY:
             return lecture_flag ? 0 : 0;
   }
}
```

Weekday	Number	Binary
Sunday	0	$(000)_2$
Monday	1	$(001)_2$
Tuesday	2	$(010)_2$
Wednesday	3	$(011)_2$
Thursday	4	$(100)_2$
Friday	5	$(101)_2$
Saturday	6	$(110)_2$

converting to a truth table

Weekday	Number	Binary
Sunday	0	$(000)_2$
Monday	1	$(001)_2$
Tuesday	2	$(010)_2$
Wednesday	3	$(011)_2$
Thursday	4	$(100)_2$
Friday	5	$(101)_2$
Saturday	6	$(110)_2$

Weekday	Lecture?	c0	c1	c2	c3
000	0	0	1	0	0
000	1	0	0	0	1
001	0	0	1	0	0
001	1	0	0	0	1
010	0	0	1	0	0
010	1	0	0	1	0
011	0	0	1	0	0
011	1	0	0	1	0
100	-	0	1	0	0
101	0	1	0	0	0
101	1	0	1	0	0
110	-	1	0	0	0
111	-	-	-	-	-

truth table ⇒ logic (part one)

DAY	d2d1d0	L	c0	c1	c2	c3
SunS	000	0	0	1	0	0
SunL	000	1	0	0	0	1
MonS	001	0	0	1	0	0
MonL	001	1	0	0	0	1
TueS	010	0	0	1	0	0
TueL	010	1	0	0	1	0
WedS	011	0	0	1	0	0
WedL	011	1	0	0	1	0
Thu	100	-	0	1	0	0
FriS	101	0	1	0	0	0
FriL	101	1	0	1	0	0
Sat	110	-	1	0	0	0
-	111	-	-	-	-	-

c3 = (DAY == SUN and LEC) or (DAY == MON and LEC)

c3 = (d2 == 0 && d1 == 0 && d0 == 0 && L == 1) ||
(d2 == 0 && d1 == 0 && d0 == 1 && L == 1)

c3 = d2'•d1'•d0'•L + d2'•d1'•d0•L

truth table ⇒ logic (part two)

c3 = d2'•d1'•d0'•L + d2'•d1'•d0•L

c2 = (DAY == TUE and LEC) or
(DAY == WED and LEC)

c2 = d2'•d1•d0'•L + d2'•d1•d0•L

DAY	d2d1d0	L	c0	c1	c2	c3
SunS	000	0	0	1	0	0
SunL	000	1	0	0	0	1
MonS	001	0	0	1	0	0
MonL	001	1	0	0	0	1
TueS	010	0	0	1	0	0
TueL	010	1	0	0	1	0
WedS	011	0	0	1	0	0
WedL	011	1	0	0	1	0
Thu	100	-	0	1	0	0
FriS	101	0	1	0	0	0
FriL	101	1	0	1	0	0
Sat	110	-	1	0	0	0
-	111	-	-	-	-	-

truth table ⇒ logic (part three)

DAY	d2d1d0	L	c0	c1	c2	c3
SunS	000	0	0	1	0	0
SunL	000	1	0	0	0	1
MonS	001	0	0	1	0	0
MonL	001	1	0	0	0	1
TueS	010	0	0	1	0	0
TueL	010	1	0	0	1	0
WedS	011	0	0	1	0	0
WedL	011	1	0	0	1	0
Thu	100	-	0	1	0	0
FriS	101	0	1	0	0	0
FriL	101	1	0	1	0	0
Sat	110	-	1	0	0	0
-	111	-	-	-	-	-

c3 = d2’•d1’•d0’•L + d2’•d1’•d0•L

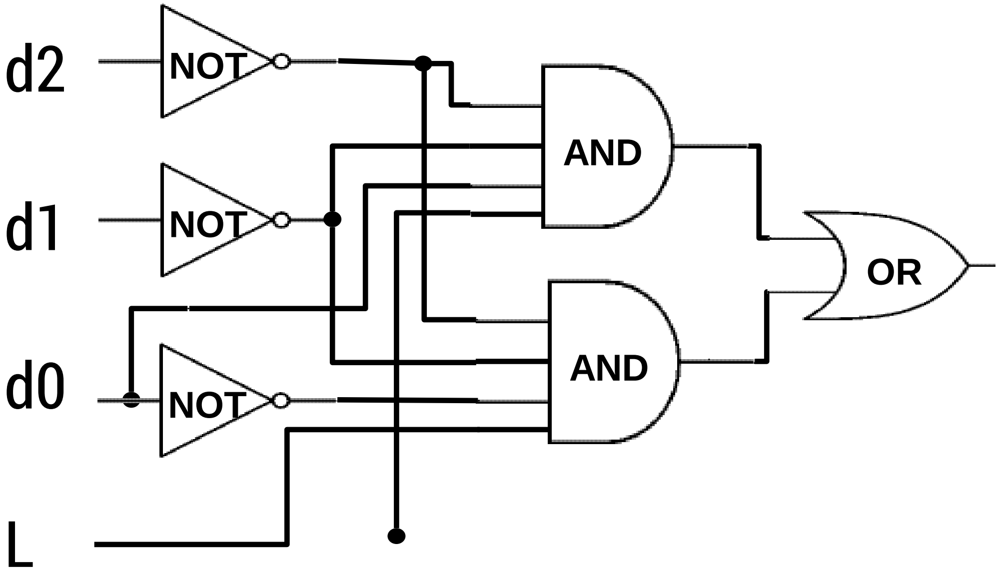
c2 = d2’•d1 •d0’•L + d2’•d1 •d0•L

c1 =

[you do this one]

c0 = d2•d1’• d0 •L’ + d2•d1 •d0’

$c3 = d2'\bullet d1'\bullet d0'\bullet L + d2'\bullet d1'\bullet d0\bullet L$



(multiple input AND gates)

[LEVEL UP]

DAY	d2d1d0	L	c0	c1	c2	c3
SunS	000	0	0	1	0	0
SunL	000	1	0	0	0	1
MonS	001	0	0	1	0	0
MonL	001	1	0	0	0	1
TueS	010	0	0	1	0	0
TueL	010	1	0	0	1	0
WedS	011	0	0	1	0	0
WedL	011	1	0	0	1	0
Thu	100	-	0	1	0	0
FriS	101	0	1	0	0	0
FriL	101	1	0	1	0	0
Sat	110	-	1	0	0	0
-	111	-	-	-	-	-