Propositional Logic

COMP1600 / COMP6260

Semester 2, 2021

This Course

Programming. (Haskell, Java, Python, . . .)

- Tell the computer *how* to perform a certain task

Logic. (

- Spe

Computation. (this course)

- the (discrete) maths of computation: what can b

You know how to program. We will explore logic to describe programs, and study fundamental (programming language independent) models of what can be computed in the first place.

Example: Vote Counting in the ACT

```
/* STEP 19 */
/* We haven't incremented count yet, so this applies to NEXT count */
mark_elected(cand, (void *)(count+1));
/* STEP 20 */
vote_value_of_surplus = cand->c[count].total - quota;
/* STEP 21 */
increment_count();
/* STEP 22 */
pile = cand->c[cand->count_when_quota_reached].pile;
non_exhausted_ballots
= (number_of_b
   - for_each_
/* STEP 23 */
if (non_exhaus
new_vote_val
else
new_vote_value = ((struct fraction) { vote_value_of_surplus,
                                                    non_exh
/* STEP 24 */
update_vote_values(pile, new_vote_value);
/* STEP 24b */
/* Report actual value (may be capped) */
report_transfer(count,
pile->ballot->vote_value,
vote_value_of_surplus);
distribute_ballots(pile, candidates,vacating);
```

(Part of the EVACS code used for vote counting in the ACT in 2012)

Do you trust this? What does it do? Does it *really* implement the law?

The two faces of Boolean logic: Circuits

"Computation with boolean / binary values: 0 and 1"

The two faces of Boolean logic: Formulae

- "Logic" as a way of reasoning
- al
- or false: T / F "

George Boole (1815 – 1864)

Truth Values

Consider *binary*, or *boolean* values

- 0, or false (written F)
- 1, or true (written T)

Boolean Functions are functions that

- tak
- pro

Example.

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

y		
F	T	T
T	F	T
T	T	F

Two Interpretations

Example.

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

x	y	$f(x,y)$
F	F	F
F	T	T
T	F	T
T	T	F

Comput

- the o
- or: co

  $x + y$ taken modulo two.

Truth of Statements

Here, x and y represent whether certain statements are true or false, e.g.:

- x could stand for "Joe was born in Melbourne"
- y could stand for "Joe was born in Sydney"

Then $f(x,y)$ is the truth value of the statement "Joe was born either in Sydney or in Melbourne"

More on the Two Interpretations

Computing with Binary Values:

- inputs are either 0 or 1
- one o
- use

Truth of Statements

- inputs describe whether certain statements ar
- output describes the truth value of a compound st
- used to describe *truth*

Building Boolean Functions

As with writing programs, boolean functions are constructed from *basic building blocks*

x AND y

x	y	x AND y
1	1	1

x OR y

x	y	x OR y
1	1	1

NOT x

x	NOT x
1	0

- AND, OR and NOT are written as $\wedge$, $\vee$ and $\neg$, respectively and called *conjunction*, *disjunction* and *negation*.
- the symbols in the last row are circuit representations

Example revisited

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

x	y	$f(x,y)$
F	F	F
F	T	T
T	F	T
T	T	F

Written

Reading of the Formula. This formula is tru i

- where either the LHS (i.e. $x \wedge \neg y$) is tru ($\neg x \wedge y$) is true, *or both* (!)
- the LHS is true if x is true and $\neg y$ is true (so y is false)
- the RHS is true if $\neg x$ is true (so x is false) and y is true
- so that the formula is true in a situation where *precisely one* of x and y are true (but not both)

About Situations

Recall. Truth of the formula $(x \quad y) \quad (\ x \quad y)$ depends on a *situation*

- tha
- can be

Definition. If V is a set of variables (e.g. above), then a situation for V is a mapping

Formula View

Q. How do we align the formula and the boolean function?

A. Construct the value of the boolean function step by step

x	y	$\neg x$	$\neg y$	$x \wedge \neg y$	$\neg x \wedge y$	$(x \wedge \neg y) \vee (\neg x \wedge y)$
F	F	T	T	F	F	F

Left two columns: all possible truth values for

- given y, we know $\neg y$ (4th column)
- given x and $\neg y$, we know $x \wedge \neg y$ (pe
- given x, we know $\neg x$ (3rd column)
- given $\neg x$ and y, we know $\neg x \wedge y$ (6th column)
- given $x \wedge \neg y$, and $\neg x \wedge y$, we know $(x \wedge \neg y) \vee (\neg x \wedge y)$ (last column)

Indeed, $(x \wedge \neg y) \vee (\neg x \wedge y)$ is exclusive-or!

Formula View, Continued

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

Q. What columns do I need?

A. Look at the target formula . . .

- to ev $\neg x \wedge y$
- to ev
- to ev

Coloured formulae indicate columns in the table.

More systematically

1. start with the target formula, and create entries f subformulae
2. repeat step 1 for all subformulae until inputs are reached.

Truth Tables

Formulae. Are constructed from T (true), F (false) and variables $(x, y, \dots)$ using $\wedge, \vee, \neg$ (more connectives later).

Truth tables for a formula

- list all T/F combinations of the variables (*all situations*)
- list t
- pos

Example. (same as before)

x	y	$\neg x$	$\neg y$	$x \wedge \neg y$	$\neg x \wedge$	
F	F	T	T	F	F	
F	T	T	F	F	T	T
T	F	F	T	T	F	T
T	T	F	F	F	F	F

On Representation

Boolean Functions with n inputs are simply functions:

$$f : \underbrace{\{0,1\} \times \cdots \times \{0,1\}}_{n \text{ times}} \to \{0,1\}.$$

Logical F

- the r
- ma

Circuits with n inputs also *represent* bool

- one boolean output is determined for each comb
- again, can have many circuits representing the same function

Summary. Boolean functions can be represented by truth tables (uniquely), formulae and circuits.

More Circuit Primitives

x XOR y

x	y	$\mathrm{xor}(x,y)$
0	0	0
0	1	1
1	0	1
1	1	0

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

x NAND y

x	y	$\mathrm{nand}(x,y)$
0	0	1
0	1	1
1	0	1
1	1	0

- we don't introduce formula type connectives for NAND and XOR as they are not commonly used (and we will not use them in what follows), but they can be encoded
- the symbols in the middle row are the circuit representations

Can We Express All Boolean Functions?

Q. Given a boolean function $f(x_1, \ldots, x_n)$, can we construct a formula that represents f?

A. Yes, this is a theorem and here is the proof. Recall that a formula represen
function t

Proof (Sk

For boolean values $i_1, \ldots, i_n \in \{T, F\}$ we have a formula ϕ such that

$$\phi_{i_1 \ldots i_n}(x_1, \ldots, x_n) = T \iff x_1 =$$

(This formula is a large $\wedge$ with elements x_ℓ if $i_\ell = T$ and $\neg x_\ell$ if $i_\ell = F$.)

Then take $\phi_f = \phi_{r_1} \vee \cdots \vee \phi_{r_k}$ where $r_1, \ldots, r_k$ are precisely the vectors $(i_1, \ldots, i_n)$ for which $f(i_1, \ldots, i_n) = T$.

Proof Detail

Suppose we have a line of the truth table

x	y	z	$f(x,y,z)$

Consider t

$$\phi_{TTF} \equiv x \wedge y \quad (\quad z)$$

then we have

$\phi_{TTF} = T$ if and only if x

Proof Detail

Now consider all lines of the truth table

x	y	z	$f(x,y,z)$
T	T	F	T
F	F	F	T
	$\cdots$		

that have a T in the right hand column, where line have a F in the right column. The formula

$$\phi_{TTF} \vee \phi_{TFF} \vee \phi_{FFF}$$

has precisely the truth table above.

Expressively Complete Sets

Definition. A set of logical connectives is *expressively complete* if it allows us to build all boolean functions.

Example. The set consisting of $\wedge$, $\vee$ and $\neg$ expressively complete.

Non-ex *not*
expressiv

Theorem. The set consisting of just nand is expressi

Proof (Sketch). Using formula notation, we ca

- express negation: $\neg x = \text{nand}(x, x)$
- express conjunction: $x \wedge y = \neg\text{nand}(x, y)$
- express disjunction: $x \vee y = \neg(\neg x \wedge \neg y)$

Elements of Counting

Q1. How many boolean functions with n inputs (and one output) can we create using AND, OR and NOT circuits / using the logical connectives $\wedge$, $\vee$, and $\neg$?

Q2. How many boolean functions with n inputs can we create *just* with OR-gates and a constant wire with value 0 / the logical connectives $\vee$ and F?

Q3. How many gates do we need in the worst case to construct a boolean function?

A Formula-Size Theorem

Theorem. Every boolean function with n input bits (and one output) can be represented using at most $10 \cdot 2^n - 10$ logical connectives from the set $\{\neg, \wedge, \vee\}$.

Proof (Sketch). We use a technique called *induction* that we will analyse more clos

Base case:

$10 \cdot 2^1 -$

Inductiv e

two $n-1$-bit functions f_0 and f_1:

$$f_0(x_2, \ldots, x_n) \equiv f(0, x_2, \ldots, x_n) \qquad f_1(x_2, \ldots, x_n)$$

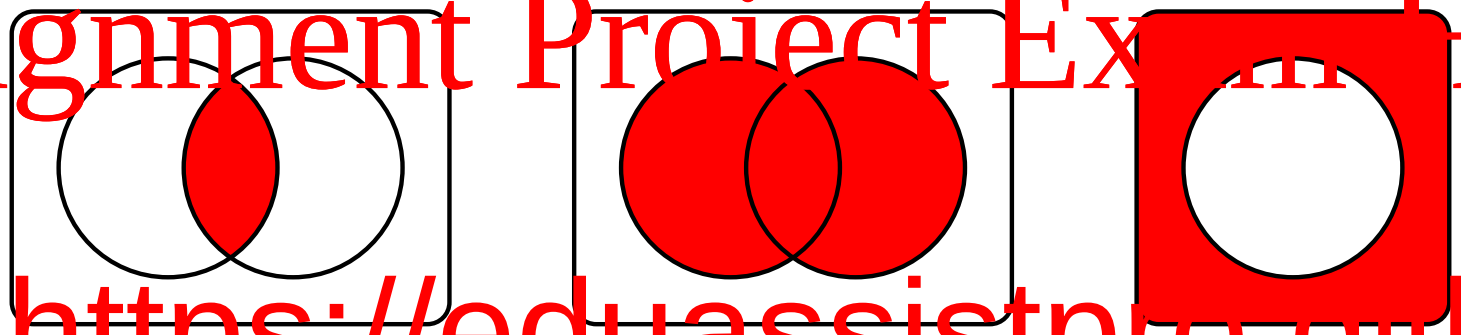
Both can be represented using at most 10

We can write $f = (\neg x_1 \wedge f_0) \vee (x_1 \wedge f_1)$. Using f_0 and f_1, we therefore need to add 4 more connectives to get a formula for f, and we have used

$$2 \cdot (10 \cdot 2^{n-1} - 10) + 4 = 10 \cdot 2^n - 20 + 4 \leq 10 \cdot 2^n - 10$$

connectives.

Visualisation: Venn Diagrams



- the boxes are the space of all situations where alse
- labelled circles describe those situations wher
- red area describes those situations where the for

(Every point in a Venn diagram describes a *situation*).

Memories from High School ...

Analogy: Algebraic Terms. Given a set V of variables, algebraic terms are constructed as follows:

- $0, 1, \ldots$ and all variables $x \in V$ are algebraic terms
- if s and t are algebraic terms, then so is $s + t$ and $s \cdot t$.

Exampl

Usual precedence. $\cdot$ binds more strongly than $+$

$x \cdot 3 + y$ reads as $(x \cdot 3) + y$

Crucial Aspect.

Terms can be *evaluated* given values for all variables.

Back to Boolean Functions

Definition. Given a set V of variables, boolean formulae are constructed as follows:

- T (true) and F (false) and all variables $x \in V$ are boolean formulae
- if ϕ and ψ are boolean formulae, then so are $\phi \wedge \psi$ and $\phi \vee \psi$.
- if ϕ is a boolean formula, then so is $\neg\phi$.

Exampl

$x))$

Precedence. $\neg$ binds more strongly than $\vee$:

$\neg x \wedge y \vee z$ reads as ((

Crucial Aspect.

Boolean formulae can be *evaluated* given (boolean) values for all variables.

Equations

Examples from Algebra.

$$x \cdot (3 + y) = x \cdot 3 + x \cdot y$$

$$25 + (18 \cdot y) = 18 \cdot y + 25$$

Boolean

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

$$T \wedge (y \vee x) =$$

Valid Equations.

For *all* values of variables, LHS and RHS evaluate to same number.

Applies to *both* algebraic terms and boolean formulae!

Valid Boolean Equations.

Associativity

$a \vee (b \vee c) = (a \vee b) \vee c$ $\qquad$ $a \wedge (b \wedge c) = (a \wedge b) \wedge c$

Commutativity

$a \vee b = b \vee a$ $\qquad$ $a \wedge b = b \wedge a$

Abs

$a \vee$ $\qquad$ a

Identity.

$a \vee F = a$ $\qquad$ a

Distributivity.

$a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$ $\qquad$ $a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c)$

Complements.

$a \vee \neg a = T$ $\qquad$ $a \wedge \neg a = F$

Equational Reasoning in Ordinary Algebra

$$
\begin{aligned}
& (x+12)\cdot 3(x+17) \\
&= (x+12)\cdot(3\cdot x+3\cdot 17) && \text{(distributivity)} \\
&= (x+12)\ \ (3\ \ x+51) \\
&= 3\cdot x\ \ +51\cdot x+36\cdot x+612 && \text{(commutativity)} \\
&= 3\cdot x^2+(51+36)\cdot x+612 && \text{(distributivity)} \\
&= 3\cdot x^2+87\cdot x+612
\end{aligned}
$$

- each step (other than addition / multiplication of numbers) justified by a "law of arithmetic"
- "pattern matching" against algebraic laws

Proving Boolean Equations

Example. We prove the law of *idempotence*:

$$\begin{aligned} x \vee x &= (x \vee x) \wedge T && \text{(identity)} \\ &= (x \vee x) \wedge (x \vee \neg x) && \text{(complements)} \end{aligned}$$

$$\begin{aligned} x & && \text{(identity)} \end{aligned}$$

Rules of Reasoning.

- All boolean equations may be assumed (with variables substituted by formulae)
- may replace formulae with formulae that are proven equal
- equality is transitive!

Two faces of boolean Equations

Truth of boolean equations:
A boolean equation $\phi = \psi$ (where ϕ, ψ are boolean formulae) is *true* if ϕ and ψ evaluate to the same truth values in all situations (i.e. for all possible tr

Equatio
A boolean equation is *provable* if it can be deriv
commutativity, absorption, identity, distributiv
the laws of equational reasoning.

Q. How do these two notions hang together?

Soundness and Completeness

Slightly Philosophical.

- *Truth* of an equation relates to the *meaning* (think: truth tables) of the connectives $\wedge$, $\vee$ and $\neg$.
- Equational *provability* relates to a method that allows us to *establish* truth of an equation.

They are

Soundn ns, then it is true.

- all basic equations (associativity, distributiv
- the rules of equational reasoning preserve truth

Completeness. If a boolean equation is true, then it is provable using equations.

- more complex proof (not given here), using the so-called *Lindenbaum Construction*.

Challenge Problem: The De Morgan Laws

De Morgan's Laws

$$\neg(x \vee y) = \neg x \wedge \neg y \qquad \neg(x \wedge y) = \neg x \vee \neg y$$

In Englis

- if it is fa
- if it is fa
  false.

Truth of De Morgan's Laws. Easy to establish via truth

Provability of De Morgan's Laws

- if the completeness theorem (that we didn't prove!) is true, then an equational proof must exist
- however, it is quite difficult to actually find it!