COMP30026 Models of Computation

Propositional Logic

Bach Le / Anna Kalenkova

Lecture Week 2 Part 1 (Zoom)

Semester 2, 2021

This Lecture is Being Recorded

Our Goal for the Next Few Lectures

- Introduce/recapitulate propositional logic.

- Use it as a vehicle for launching more generally applicable logic concepts.
- Use it for simple, mechanised reasoning.

If you are familiar with propositional logic, some of this will be old hat.

But pay attention anyway, because the concepts and methods we introduce now will serve as a blueprint for similar (but more complex and powerful) concepts and methods for predicate logic.

Administrivia: Grok

Solutions to module 1 problems are now available—Study them!
The first serious assessment tasks are not far away.

Do not neglect Grok! We need to see that your account is working.

Of all Grok enrolled students, 11 have completed all exercises in all 8 modules, and 46 have completed the 6 mandatory modules.

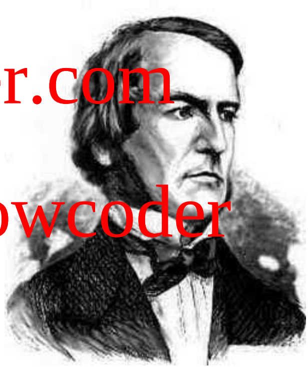
415 have at least one green diamond. But 105 have not completed module 1 yet. And more than 60 are still to do their first exercise!

If you don't have a Grok account, please identify yourself asap.

Propositional = Boolean Logic

Philosophers have been interested in the "rules of reasoning" for thousands of years. Aristotle's syllogisms had particular importance for European scholars.

George Boole is usually considered the father of modern logic. Boole took an algebraic view of logic, pointing out that there are important abstract analogies between certain arithmetic operations and the logical connectives.



Intro Puzzle

Huey, Dewey and Louie are being questioned by their uncle. Here is what they say:

> *Huey: "Dewey and Louie had equal share in it; if one is guilty, so is the other."*
>
>
>
> *Dewey: "If Huey is guilty, then so am I."*
>
> *Louie: "Dewey and I are not both guilty."*

Their uncle, knowing that they are cub scouts, realises that they cannot tell a lie.

Has he got sufficient information to decide who (if any) are guilty?

(Classical) Propositional Logic: Syntax

We shall build propositional formulas from this set of symbols:

$$\underbrace{A, B, C, \ldots Z}_{\text{prop. letters}}, \underbrace{\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow, \oplus}_{\text{connectives}}, \mathbf{f}, \mathbf{t}, (,).$$

Well-formed formulas (wffs) are generated by the grammar

$$\begin{array}{lcl}
wff & \to & A \mid B \mid C \mid \ldots \mid Z \mid \mathbf{f} \mid \mathbf{t} \\
 & \mid & (\neg wff) \\
 & \mid & (wff \wedge wff) \\
 & \mid & (wff \vee wff) \\
 & \mid & (wff \Rightarrow wff) \\
 & \mid & (wff \Leftrightarrow wff) \\
 & \mid & (wff \oplus wff)
\end{array}$$

Propositional Logic: Notational Conveniences

We shall drop outermost parentheses.

We shall assume that $\neg$ binds tighter than $\wedge$ and $\vee$.

These bind tighter than $\oplus$, which binds tighter than $\Rightarrow$ and $\Leftrightarrow$.

This allows us to write, without ambiguity,

$$(((\neg Q) \wedge P) \Rightarrow (P \vee (P \Leftrightarrow Q)))$$

as

$$\neg Q \wedge P \Rightarrow P \vee (P \Leftrightarrow Q)$$

Note: O'Donnell et al. (and Makinson) use $\rightarrow$ instead of $\Rightarrow$, and $\leftrightarrow$ instead of $\Leftrightarrow$. Makinson also uses 0 for **f** and 1 for **t**. On a whiteboard I tend to use 0 and 1, as they are faster to write.

Propositional Logic: Semantics

A proposition is false (**f**) or true (**t**).

A truth assignment maps each propositional letter to **t** or **f**.

We can give the semantics of the connectives via truth tables:

A	B	$\neg A$	$A \wedge B$	$A \vee B$	$A \Rightarrow B$	$A \Leftrightarrow B$	$A \oplus B$
f	**f**	**t**	**f**	**f**	**t**	**t**	**f**
f	**t**	**t**	**f**	**t**	**t**	**f**	**t**
t	**f**	**f**	**f**	**t**	**f**	**f**	**t**
t	**t**	**f**	**t**	**t**	**t**	**t**	**f**

This gives meaning to all propositional formulas, as we let A and B stand for the values of arbitrary (compound) propositions.

Connectives Defined in Haskell

Haskell has a type Bool, and some connectives are pre-defined:


```
data Bool = False | True

not :: Bool -> Bool
not True  = False
not False = True

(&&) :: Bool -> Bool -> Bool
False && _ = False
True  && x = x

(||) : Bool -> Bool -> Bool
False || x = x
True  || _ = True
```

Conjunction and Disjunction

$P \wedge Q$ is the conjunction of P and Q.

$P \vee Q$ is their disjunction.

An "or" in English sometimes translates to disjunction:

I'll eat if there is peanut butter or jam in the fridge.

Other times it translates to exclusive or:

Would you like the ice cream or the crème brûlée?

The Conditional

The proposition $P \Rightarrow Q$ is best read "if P then Q" (or sometimes "P only if Q" or "Q whenever P"). Usually, "implies" is misleading.

A	B	$A \Rightarrow B$
f	**f**	**t**
f	**t**	**t**
t	**f**	**f**
t	**t**	**t**

1. If the volume is increased, the pressure falls.

The Conditional

The proposition $P \Rightarrow Q$ is best read "if P then Q" (or sometimes "P only if Q" or "Q whenever P"). Usually, "implies" is misleading.

A	B	$A \Rightarrow B$
f	**f**	**t**
f	**t**	**t**
t	**f**	**f**
t	**t**	**t**

1. If the volume is increased, the pressure falls.
2. If Melbourne is in Queensland then Brisbane is in Victoria.

The Conditional

The proposition $P \Rightarrow Q$ is best read "if P then Q" (or sometimes "P only if Q" or "Q whenever P"). Usually, "implies" is misleading.

A	B	$A \Rightarrow B$
f	**f**	**t**
f	**t**	**t**
t	**f**	**f**
t	**t**	**t**

1. If the volume is increased, the pressure falls.
2. If Melbourne is in Queensland then Brisbane is in Victoria.
3. Melbourne and Brisbane are in different states and if Melbourne is in Queensland then so is Brisbane.

We talk about material implication.

Note that $A \Rightarrow B$ has the same truth table as $\neg A \vee B$.

More Connectives in Haskell


```
infix 0 ==>
infix 0 <=>
infix 1 <+>

(==>) :: Bool -> Bool -> Bool
False ==> _ = True
True  ==> x = x

(<=>) :: Bool -> Bool -> Bool
x <=> y  =  x == y

(<+>) :: Bool -> Bool -> Bool
x <+> y  =  x /= y
```

Poll 1

Which of these claims hold?

1. $P \Rightarrow Q$ has the same truth table as $\neg Q \Rightarrow \neg P$
2. $(P \Rightarrow Q) \wedge (P \Rightarrow R)$ has the same truth table as $P \Rightarrow (Q \wedge R)$
3. $(P \Rightarrow R) \wedge (Q \Rightarrow R)$ has the same truth table as $(P \wedge Q) \Rightarrow R$

Other Binary Connectives

We can also define $\downarrow$, or "nor", as well as $\uparrow$, or "nand".

A	B	$A \downarrow B$	$A \uparrow B$
f	**f**	**t**	**t**
f	**t**	**f**	**t**
t	**f**	**f**	**t**
t	**t**	**f**	**f**

"Nand" is sometimes called Sheffer's stroke.

Some Ternary Connectives

A	B	C	if A then B else C	$median(A, B, C)$
f	**f**	**f**	**f**	**f**
f	**f**	**t**	**t**	**f**
f	**t**	**f**	**f**	**f**
f	**t**	**t**	**t**	**t**
t	**f**	**f**	**f**	**f**
t	**f**	**t**	**f**	**t**
t	**t**	**f**	**t**	**t**
t	**t**	**t**	**t**	**t**

On Boolean Short-Circuit Definitions

Most programming languages offer the Boolean connectives ‘and’ and ‘or’, but usually these are not commutative!

In C, Haskell, and many other languages, `0 == 1 && 1/0 == 42` has a behaviour that is different from `1/0 == 42 && 0 == 1`.

One evaluates to `False`, the other causes a run-time error. The first version avoids the runtime error, because conjunction is not a strict function in typical programming languages: If the first argument is false, the second won’t be evaluated.

To model the behaviour properly, we really need three-valued propositional logic, the third truth value being “undefined”.

Knights and Knaves Puzzle

On the island of Knights and Knaves, everyone is a knight or knave. Knights always tell the truth. Knaves always lie.

Today there is a census on the island!

You are a census taker, going from house to house. Fill in what you know about each of these three houses.

- **In house 1:** Husband: We are both knaves.

Knights and Knaves Puzzle

On the island of Knights and Knaves, everyone is a knight or knave. Knights always tell the truth. Knaves always lie.

Today there is a census on the island!

You are a census taker, going from house to house. Fill in what you know about each of these three houses.

- **In house 1:** Husband: We are both knaves.
- **In house 2:** Wife: At least one of us is a knave.

Knights and Knaves Puzzle

On the island of Knights and Knaves, everyone is a knight or knave. Knights always tell the truth. Knaves always lie.

Today there is a census on the island!

You are a census taker, going from house to house. Fill in what you know about each of these three houses.

- **In house 1:** Husband: We are both knaves.
- **In house 2:** Wife: At least one of us is a knave.
- **In house 3:** Husband: If I am a knight then so is my wife.