CO140 - Logic

Material Order

Seeing as this module isn't on Panopto, these notes are solely based off of the provided notes on CATe. This is the order in which they are uploaded (and I'd assume the order in which they are taught).

1. *Propositional Logic - Syntax.pdf*
2. *Propositional Logic - Semantics.pdf*
3. *Propositional Logic - English Correspondence.pdf*
4. *Propositional Logic - Arguments and Validity.pdf*
5. *PropositionalLogic - CheckValidity.pdf*
6. *Propositional Logic - Natural Deduction Part1.pdf*
7. *Propositional Logic - Natural Deduction Part 2.pdf*
8. *Propositional Logic- Natural Deduction Part 3.pdf*
9. *Propositional Logic - Lemmas.pdf*
10. *First-order logic.pdf*

Introduction

A logic system consists of 3 things:

1. Syntax - formal language used to express concepts
2. Semantics - meaning for the syntax
3. Proof theory - syntactic way of identifying valid statements of language

Considering the basic example in a program, we can then see the features;

```
if count > 0 and not found then
    decrement count;
    look for next entry;
end if
```

1. basic (**atomic**) statements (**propositions**) are either $\top$ or $\bot$ depending on circumstance;
   i. `count > 0`
   ii. `found`
2. **boolean operations**, such as `and`, `or`, `not`, etc. are used to build complex statements from **atomic propositions**
3. the final statement `count > 0 and not found` evaluates to either $\top$ or $\bot$

Syntax

The formal language of logic consists of three ingredients;

1. Propositional atoms (propositional variables), evaluate to a truth value of either $\top$ or $\bot$. These are represented with letters; $p, p', p_0, p_1, p_2, p_n, q, r, s, ...$

2. Boolean connectives;

• `and` is written as $p \wedge q$	p and q both hold
• `or` is written as $p \vee q$	p or q holds (or both)
• `not` is written as $\neg p$	p does not hold
• `if-then / implies` is written as $p \to q$	if p holds, then so does q
• `if-and-only-if` is written as $p \leftrightarrow q$	p holds if and only if q holds
• **`truth`**, and **`falsity`** are written as $\top$, and $\bot$ respectively.	logical constants

3. Punctuation. Similar to arithmetic, the lack of brackets can make an expression ambiguous. For example, $p_0 \vee p_1 \wedge p_2$ can be read as either $(p_0 \vee p_1) \wedge p_2$ or $p_0 \vee (p_1 \wedge p_2)$, which are different. The latter is the correct interpretation due to binding conventions.

   We can order the boolean connectives by decreasing binding strength;

   (strongest) $\neg$, $\wedge$, $\vee$, $\to$, $\leftrightarrow$ (weakest)

   While repeated disjunctions ($\vee$), and conjunctions ($\wedge$) are fine, as $p \wedge q \wedge r$ is equivalent to $p \wedge (q \wedge r)$, and the same for $\vee$, due to associativity, the same isn't true for $\to$. Due to the ambiguity, brackets should always be used.

   There are also exceptions to the rule, for example with $p \to r \wedge q \to r$ - this should be $p \to (r \wedge q) \to r$ according to our binding conventions, but brackets should be used to ensure the correct interpretation.

Formulas

Something is a **well-formed formula** only if it is built from the following rules (the brackets are required);

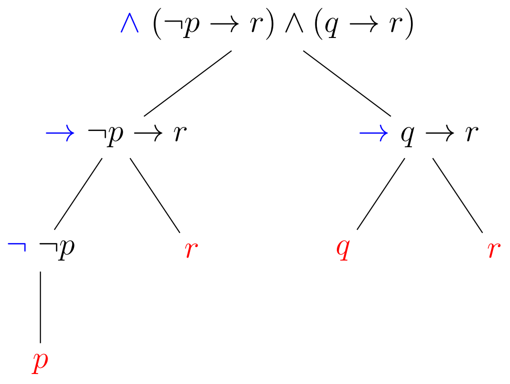
1. a propositional atom $(p, p', p_0, p_1, p_2, p_n, q, r, s, ...)$ is a propositional formula

2. $\top$, and $\bot$ are both formulas

3. if A is a formula, then $(\neg A)$ is also a formula

4. if A, and B are both formulas, then $(A \wedge B), (A \vee B), (A \to B), (A \leftrightarrow B)$ are also formulas

We can also create a tree to parse a logical formula, for example; $(\neg p \to r) \wedge (q \to r)$



Note that this tree shows the principal connective in blue, and the propositional atoms in red. Note that $\wedge$ is the principal connective in the top layer, and it therefore has the general form $A \wedge B$, and so on going down.

Definitions

- A formula is a **negated formula** when it is in the form $\neg A$, negated atoms are sometimes called **negated-atomic**.
- $A \wedge B$, and $A \vee B$ are **conjunctions**, and **disjunctions**. A, and B, are **conjuncts**, and **disjuncts**, respectively.
- $A \rightarrow B$ is an implication. A is the **antecedent**, and B is the **consequent**

Semantics

The connectives covered above have a rough English translation. However a natural language has ambiguity, and as engineers, we need precise meanings for formulas. This is the truth table for every connective that will be used in this course (?):

p	q	$\top$	$\bot$	$p \wedge q$	$p \vee q$	$\neg p$	$p \rightarrow q$	$p \leftrightarrow q$	$p \uparrow q$
0	0	1	0	0	0	1	1	1	1
0	1	1	0	0	1	1	1	0	1
1	0	1	0	0	1	0	0	0	1
1	1	1	0	1	1	0	1	1	0

Note how we can also define new connectives (see how $A \uparrow B$ was defined in the last column); this is a NAND connective - equivalent to $\neg(A \wedge B)$.

Translation

English to Logic

- **but** means `and`

  "I will go out, but it is raining" `(i will go out)` $\wedge$ `(it is raining)`

- **unless** generally means `or`

  "I will go out unless it rains" `(i will go out)` $\vee$ `(it will rain)` (note the `will`)

  $\neg$`(it will rain)` $\rightarrow$ `i will go out`

  There is also the strong form of **unless**, but in we generally use the weak form in computing

  `(i will go out)` $\leftrightarrow$ $\neg$`(it will rain)`

- **or** generally refers to exclusive or (strong reading) in English, but it can also refer to inclusive or (weak reading). However, we always take the weak reading in computing.

Modality

`I don't know what this means, so I'm just ignoring it for now`

Logic to English

While the others are slightly more straightforward, $\rightarrow$ is a pain to translate.

For example, `(i am the pope)` $\rightarrow$ `(i am an atheist)` evaluates to true, as falsity implies anything, however if we were to translate it into English, "If I am the Pope, then I am an atheist" is (most likely) untrue.

Another example is the following; $p \wedge q \rightarrow r$, and $(p \rightarrow r) \vee (q \rightarrow r)$ are logically equivalent, but can be translated into different meanings. For example, let p be "event A happens", let q be "event B happens", and q be "event C happens". The former can be translated to "If both A and B happens, then C happens", whereas the latter becomes "If A happens, then C happens, or if B happens, then C also happens".

Arguments

We use the double turnstile, $\vDash$ (`\vDash` in LaTeX), to mean **therefore**. For example, the *Socrates syllogism* can be expressed as `(socrates is a man)`, `(men are mortal)` $\vDash$ `(socrates is mortal)` in logic, and in English as;

- Socrates is a man
- Men are mortal
- Therefore, Socrates is mortal

The definition of a valid argument is as follows;
Given valid formulas $A_1, A_2, ..., A_n, B$, and '$A_1, ..., A_n$ therefore B', we can write it as $A_1, ..., A_n \vDash B$, iff B is true in every situation where $A_1, ..., A_n$ are all true.

Examples

- $A, A \to B \vDash B$ **modus ponens**
- $A \to B, \neg B \vDash \neg A$ **modus tollens**
- $A \to B, B \nvDash A$ A can be false, as falsity implies anything

Definitions

- A propositional formula is logically **valid** if it's true in all situations ($\vDash A$), if A is **valid**
- A propositional formula is **satisfiable** if it's true in at least one situation (hence **valid** $\to$ **satisfiable**)
- Two propositonal formulas are logically **equivalent** if they are true in the same situations.

argument	validity	satisfiability	equivalence
$A \vDash B$	$A \to B$ valid	$A \wedge \neg B$ unsatisfiable	$(A \to B) \equiv \top$
$\top \vDash A$	A valid	$\neg A$ unsatisfiable	$A \equiv \top$
$A \nvDash \perp$	$\neg A$ not valid	A satisfiable	
$A \vDash B$, and $B \vDash A$		$A \leftrightarrow \neg B$ unsatisfiable	$A \equiv B$

(copied directly from *Propositional Logic - Arguments and Validity.pdf*)

Validity

The main ways used to check validity are as follows;

- Truth tables - check all possible situations, and check the results of each formula are $\top$
- Direct argument
- Equivalences - using equivalences to reduce the initial formula to $\top$
- Various proof systems - including Natural Deduction

In general, if we want to show that A is logically equivalent to B, we need to show $A \leftrightarrow B$ is **valid**.

Truth Tables

The use of truth tables to prove validity is fairly self-explanatory; as we're testing each situation, it's the easiest method (and it works for propositional logic since we have a finite number of configurations - doesn't work for first-order), however it's inelegant, and quite tedious depending on the number of propositional atoms.

For example, if we were to prove $(p \to q) \leftrightarrow (\neg p \vee q)$ is valid, we have to evaluate all of the subformulas.

p	q	$p \to q$	$\neg p$	$\neg p \vee q$	$(p \to q) \leftrightarrow (\neg p \vee q)$
0	0	1	1	1	1
0	1	1	1	1	1
1	0	0	0	0	1
1	1	1	0	1	1

(copied directly from *Propositional Logic - CheckValidity.pdf*)

As the propositional formula has 1 in all four possible configurations of p, and q, we can then say it is valid, and as such, $p \to q$ is logically equivalent to $\neg p \vee q$

Direct Argument

We can show the validity of $((p \to q) \to p) \to p$ (known as *Peirce's law*) with direct argument.

We can take an argument by cases, either p is $\top$ or p is $\bot$.

- $p \leftrightarrow \top$ - we know this is true as $A \to B$ is $\top$ whenever B is $\top$
- $p \leftrightarrow \bot$ - we have $p \to q$ evaluating to $\top$, as $A \to B$ is $\top$ whenever A is $\bot$. As such, this formula is evaluated to $(\top \to p) \to q$. However, we know that p is $\bot$, hence we have $\top \to \bot$, which we know evaluates to $\bot$ by the truth table for $\to$. As such, we have $\bot \to p$, hence it follows that it is valid, seeing as $A \to B$ is $\top$ whenever A is $\bot$.
- This is an argument by cases, known as **law of excluded middle** (you will use this often in Natural Deduction).

Equivalences

Refer to *Logic cribsheet.pdf* for a full list of equivalences

1. $A \wedge B \equiv B \wedge A$ — commutativity of $\wedge$
2. $A \wedge A \equiv A$ — idempotence of $\wedge$
3. $A \wedge \top \equiv A$
4. $\bot \wedge A, \neg A \wedge A \equiv \bot$
5. $(A \wedge B) \wedge C \equiv A \wedge (B \wedge C)$ — associativity of $\wedge$
6. $A \vee B \equiv B \wedge A$ — commutativity of $\vee$
7. $A \vee A \equiv A$ — idempotence of $\vee$
8. $\bot \vee A \equiv A$
9. $\top \vee A, \neg A \vee A \equiv \top$
10. $(A \vee B) \vee C \equiv A \vee (B \vee C)$ — associativity of $\vee$
11. $\neg\top \equiv \bot$

12. $\neg\bot \equiv \top$

13. $\neg\neg A \equiv A$

14. $A \to A \equiv \top$

15. $\top \to A \equiv A$

16. $A \to \top \equiv \top$

17. $\bot \to A \equiv \top$

18. $A \to \bot \equiv \neg A$

19. $A \to B \equiv \neg A \lor B$

20. $A \leftrightarrow B \equiv (A \to B) \land (B \to A) \equiv (A \land B) \lor (\neg A \land \neg B) \equiv \neg A \leftrightarrow \neg B$

21. $\neg(A \leftrightarrow B) \equiv \neg A \leftrightarrow B \equiv ...$ the rest can be derived from the above

22. $\neg(A \land B) \equiv \neg A \lor \neg B$ de Morgan laws

23. $\neg(A \lor B) \equiv \neg A \land \neg B$ de Morgan laws

24. $A \land (B \lor C) \equiv (A \land B) \lor (A \land C)$

25. $A \lor (B \land C) \equiv (A \lor B) \land (A \lor C)$

26. $A \lor (A \land B) \equiv A \lor (A \land B) \equiv A$

Normal Forms

- A formula is in **disjunctive** NF (**DNF** - $\lor$) if it's a disjunction of conjunctions of literals
- A formula is in **conjunctive** NF (**CNF** - $\land$) if it's a conjunction of disjunctions of literals (a conjunction of clauses)

Rewriting

1. Get rid of $\to$, and $\leftrightarrow$

   Replace $A \to B$ with $\neg A \lor B$

   Replace $A \leftrightarrow B$ with $(A \land B) \lor (\neg A \land \neg B)$

2. Use de Morgan laws to push negations down to the atoms

3. Delete double negations (replace $\neg\neg A$ with A)

4. Rearrange with distributivity equivalences to the desired form

5. Use the equivalences which reduce two atoms to one ($\bot \lor A \equiv A$ etc.) until no further progress can be made

Example

Write $p \land q \to \neg(p \leftrightarrow \neg r)$ in DNF

1. $p \land q \to \neg(p \leftrightarrow \neg r)$
2. $\neg(p \land q) \lor \neg(p \leftrightarrow \neg r)$ — remove $\to$
3. $\neg(p \land q) \lor \neg((p \land \neg r) \lor (\neg p \land r))$ — remove $\leftrightarrow$
4. $\neg p \lor \neg q \lor \neg(p \land \neg r) \land \neg(\neg p \land r)$ — de Morgan
5. $\neg p \lor \neg q \lor (\neg p \lor r) \land (p \lor \neg r)$ — de Morgan
6. $\neg p \lor \neg q \lor ((\neg p \lor r) \land p) \lor ((\neg p \lor r) \land \neg r)$ — distributivity of $\land$
7. $\neg p \lor \neg q \lor (\neg p \land p) \lor (r \land p) \lor (\neg p \land \neg r) \lor (r \land \neg r)$ — distributivity of $\land$
8. $\neg p \lor \neg q \lor (r \land p) \lor (\neg p \land \neg r)$ — distributivity of $\land$
9. $\neg p \lor \neg q \lor (r \land p)$ — $A \lor (A \land B) \equiv A$

   While this is in DNF, we can leave it, and simplify further

10. $\neg q \lor ((r \lor \neg p) \land (p \lor \neg p))$ — distributivity of $\lor$
11. $\neg q \lor r \lor \neg p$ — $A \land (B \lor \neg B) \equiv A$ (combination of equivalences)

Natural Deduction

```
Read Jordan Spooner's notes for this; just practice it on Pandora until you feel
confident.  These are just some key points from the slides, examples are excluded,
as it's better to just do questions
```

- If we prove A, and B, we get $A \land B$ ($\land$I)
- If we have $A \land B$, we get both A, and B ($\land$E)
- We often need to make a temporary assumption, for example, if we assume A, and in that box, we get B, then it follows that $A \to B$ ($\to$I)

  nothing in the box can be used outside of it; consider it as a scope
- If we have $A \to B$, and A, we get B ($\to$E)
- If we have A, we get $A \lor B$ ($\lor$I)

  it doesn't matter what B is, it can literally be $\bot$
- If we have $A \lor B$, and we can prove $A \to C$, and $B \to C$ (see in the *Translation* section for a similar example to $(p \lor q) \to r \equiv (p \to r) \land (q \to r)$), we get C ($\lor$E).
- Note that $\vdash$, and $\vDash$, are different. The former is syntactic, and involves proofs, whereas the latter is semantic, and involves situations
- If assuming A leads to $\bot$, we get $\neg A$ ($\neg$I)
- If assuming $\neg A$ leads to $\bot$, we get A ($\neg$E)
- If we have $\neg\neg A$, we get A ($\neg\neg$)
- If we have both A, and $\neg A$, we get $\bot$ ($\bot$I)

- If we have $\bot$, we get A ($\bot$E)

  A can be anything, as we can prove anything from $\bot$

- If we prove $A \to B$, and $B \to A$, we get $A \leftrightarrow B$ ($\leftrightarrow$I)

- If we prove $A \leftrightarrow B$, and A (or B), we get B (or A), respectively ($\leftrightarrow$E)

- PC is a derived rule from $\neg$I, and $\neg\neg$

Definitions

- If a formula can be proved by a given proof system, it's a **theorem** (hence a **theorem** is any formula A where $\vdash A$)

- A system is **sound** if every theorem is valid, and **complete** if every valid formula is a theorem

- A formula is **consistent** if $\nvdash \neg A$

- A formula is consistent iff it is satisfiable

First-order Predicate Logic

While we can use direct argument, equivalences, and natural deduction, truth tables can no longer be used, since we're working on an infinite set of possible situations.

Limitations of Propositional Logic

- the list is ordered

- every worker has a boss

- there is someone worse off than you

- it also can't express some of de Morgan's arguments, for example;

  i. a horse is an animal

  ii. therefore, the head of a horse is the head of an animal

Splitting the Atom

Previously, we considered phrases such as `the left lift is falling`, and `Adam gets in the left lift`, as atomic, without any internal structure. However, we can then regard `falling` as a **property**, or an **attribute**.

- a **unary** relation symbol takes one argument, hence it has an **arity** of 1.

  e.g. `falling(LeftLift)`

- a **binary** relation symbol takes two arguments, hence it has an arity of 2.

  e.g. `gets_in(Adam, LeftLift)`

- **constants**, which can name individual objects

  e.g. `LeftLift`, or `Adam`

While `gets_in(Adam, LeftLift)` doesn't seem that different from the original propositional atom `Adam gets in the left lift`, predicate logic is able to vary the arguments passed into the relation `gets_in`. The machinery used in predicate logic is **quantifiers**. In first-order logic, we have two quantifiers;

- $\forall$ - 'for all' e.g $\forall x(A)$, means that the predicate A applies to all x.
- $\exists$ - 'exists' (or 'some') e.g. $\exists x(A)$, means that the predicate A applies to at least one x.

Expressions like `LeftLift`, or `Adam` are constants, but to express more complex statements we need stuff like;

- $\exists x(\texttt{falling}(x) \wedge \texttt{gets\_in}(\texttt{Adam}, x))$ `Adam` gets into a falling lift
  Some x is a falling lift, and `Adam` gets in x
- $\exists x(\texttt{falling}(x))$ There exists an x that is a falling lift
- $\forall x(\texttt{falling}(x))$ Everything is a falling lift

Signatures

A **signature** is a set of constants, and relation symbols with specific arities. Also known as **similarity type**, **vocabulary**, or **language** (loosely)

This replaces the collection of propositional atoms we previously used in propositional logic. Usually L denotes a signature, $c, d, ...$ for constants, and $P, Q, R, S, ...$ for relation symbols.

Let us define an example signature L, consisting of the following;

- constants
  - `Adam`
  - `Ben`
  - `Charlie`
  - `Apple`
  - `Orange`
  - `Kale`
  - `Phone`
- unary relations (arity 1)
  - `fruit`
  - `human`
  - `student`
- binary relations (arity 2)
  - `ate`

Everything listed in L are just symbols, hence they don't come with any meaning. We will need to add a **situation**.

Terms

In order to write formulas, we need **terms** to name objects, they are not true or false, since they themselves are not formulas.

With a fixed signature L, any constant in L is an L-term, as well as any variable. Nothing else is an L-term.

A **closed** (or **ground**) term doesn't involve a variable. Hence constants are **ground** terms, and variables are not.

Formulas

Once again, with a fixed signature L, we can say the following are formulas, and nothing else is;

- given an n-ary relation R in L, and a set of L-terms $(t_1, t_2, ..., t_n)$, then $R(t_1, t_2, ..., t_n)$ is an atomic L-formula
- if t, and t' are L-terms, then $t = t'$ is an atomic L-formula (equality)
- $\top$, and $\bot$, are atomic L-formulas
- if A, and B are L-formulas, then so are $(\neg A)$, $(A \wedge B)$, $(A \vee B)$, $(A \to B)$, and $(A \leftrightarrow B)$
- if A is an L-formula, then $\forall x(A)$, and $\exists x(A)$ are L-formulas

Note that the binding conventions are the same as propositional logic, with the additional fact that $\forall x$, and $\exists x$ have the same binding strength as $\neg$

Now that we have these definitions, we can begin to construct examples of first-order logical formulas;

$\texttt{ate}(\texttt{Adam}, x)$	Adam ate x
$\exists x(\texttt{ate}(\texttt{Adam}, x))$	Adam ate something
$\forall x(\texttt{student}(x) \to \texttt{human}(x))$	all students are human (important)
$\forall x(\texttt{ate}(\texttt{Adam}, x) \to \texttt{fruit}(x))$	Adam only ate fruits / Everything Adam ate is a fruit
$\forall x \exists y(\texttt{ate}(x, y))$	eveyone ate something
$\exists y \forall x(\texttt{ate}(x, y))$	there is something that everyone ate
$\exists x \forall y(\texttt{ate}(x, y))$	someone ate everything

Note the subtle differences in the latter three examples, and how they have a rather drastic impact on the meaning of the formula.

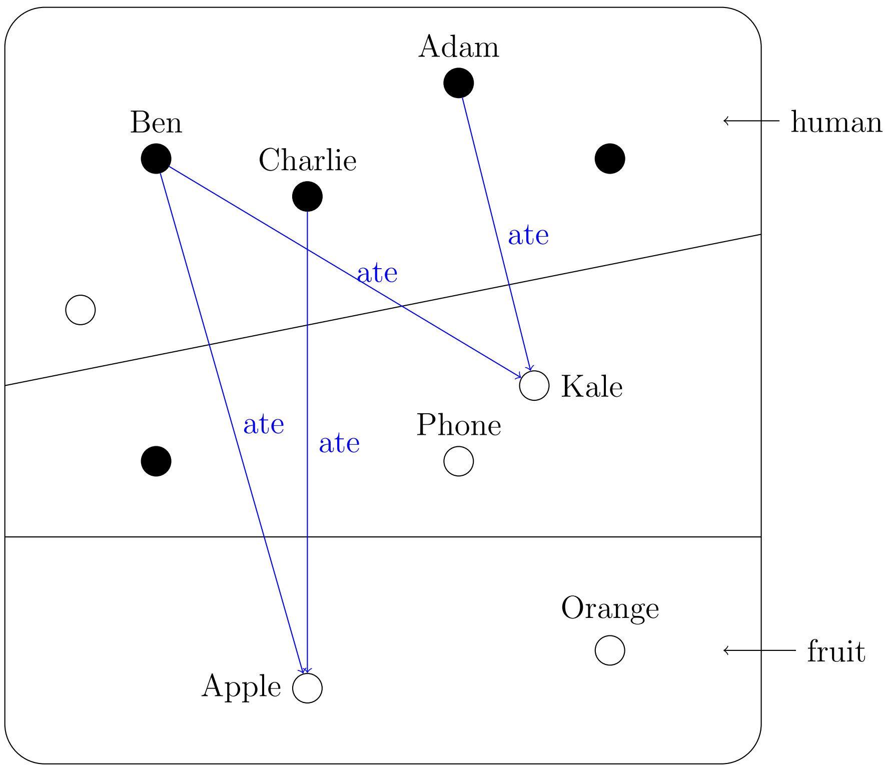
Semantics

Like in propositional logic, we have to specify a **situation** for predicate logic, and how to evaluate predicate logic formulas in a specific situation.

With a given signature L, we have an L-structure (or **model**) M, which identifies an non-empty set of objects that M covers. It is defined as the **domain**, or **universe** of M, also known as $\text{dom}(M)$. M also specifies the meanings of the symbols in L, in terms of the objects in $\text{dom}(M)$.
An **object** in $\text{dom}(M)$ is the interpretation in M of a constant, and **relation** on $\text{dom}(M)$ is the interpretation in M of a relation symbol.

Using the our previously defined symbols, we can define a model M on L, which must state which objects are in $\text{dom}(M)$, which objects are the constants (`Adam`, `Ben`, ...), which objects are `human`, `student`, `fruit`, and which objects `ate` which.

- the labelled nodes represent the constants in L
- the interpretations / meanings of `fruit`, and `human` are drawn as regions (the arrows)
- the interpretation of `student` are the black nodes
- the interpretation of the binary relation `ate` is shown by the arrow between two nodes



NOTATION: given an L-structure M, and a constant c in L, we use the notation c^M to denote the interpretation of c in M. c is an object in dom(M) that c names in M. Therefore the black node in the model, is $\texttt{Adam}^M$, which is not to be confused with `Adam`. The meaning of a constant c is the object c^M, which is assigned by the L-structure M, hence a constant can have multiple meanings since each L-structure assigns c a new meaning.

While our model is quite simple, it illustrates the basic requirements for an L-structure; it has the collection of objects (dom(M)), marks the constants, marks which objects satisfy the unary relations, as well as directed arrows showing which pairs of objects satisfy the binary relations. Generally, there isn't any easy way to represent n-ary relations (where $n \geq 3$). 0-ary (nullary) relations are propositional atoms.

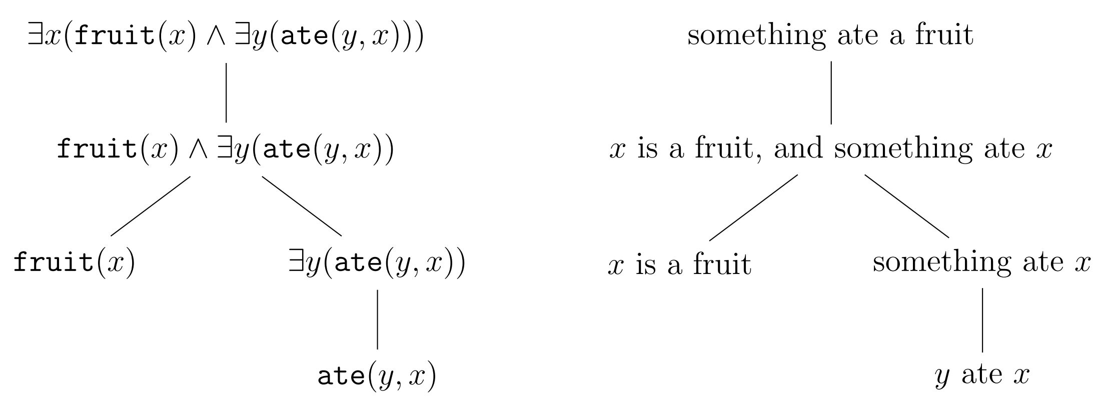
The structure M tells us that `student(Ben)` is true, as the node representing $\texttt{Ben}^M$ is coloured black; therefore this can be written as $M \vDash \texttt{student(Ben)}$ - or M says `student(Ben)`. M also tells us that `ate(Ben, Orange)` is false, hence we can write $M \nvDash \texttt{ate(Ben, Orange)}$. M also states $\texttt{Kale}^M$ is not `human`, therefore we are able to write $M \vDash \neg\texttt{human(Kale)}$.

This is a different use of $\vDash$ from the start of the module.

`There should be a section here about another model on the same signature, but that takes too much effort to draw.`

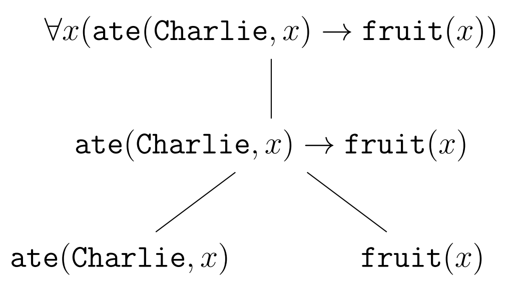
TIP: if you have something asking $M \vDash \forall x(R(x, ...) \to B)$?, the idea is to restrict the $\forall x$. We know that falsity implies anything from propositional logic, and therefore we only need to consider the cases in which $R(x, ...)$ is true. For example, on M, if we were asked $M \vDash \forall x(\texttt{ate}(\texttt{Adam}, x) \to \texttt{ate}(\texttt{Ben}, x))$?, instead of evaluating all the objects in dom(M), we should only consider the ones in which $\texttt{ate}(\texttt{Adam}, x)$ evaluates to true, which would be just the object $\texttt{Kale}^M$. And as $\texttt{ate}(\texttt{Ben}, \texttt{Kale}^M)$ is true, the statement is valid.

TIP: for a fairly complex formula (we'll work with $\exists x(\texttt{fruit}(x) \wedge \exists y(\texttt{ate}(y, x)))$), a simple method is to work out what each subformula says in English (working upwards from the atomic subformulas). For example;


$\exists x(\texttt{fruit}(x) \land \exists y(\texttt{ate}(y, x)))$
$\texttt{fruit}(x) \land \exists y(\texttt{ate}(y, x))$
$\texttt{fruit}(x)$
$\exists y(\texttt{ate}(y, x))$
$\texttt{ate}(y, x)$
something ate a fruit
x is a fruit, and something ate x
x is a fruit
something ate x
y ate x

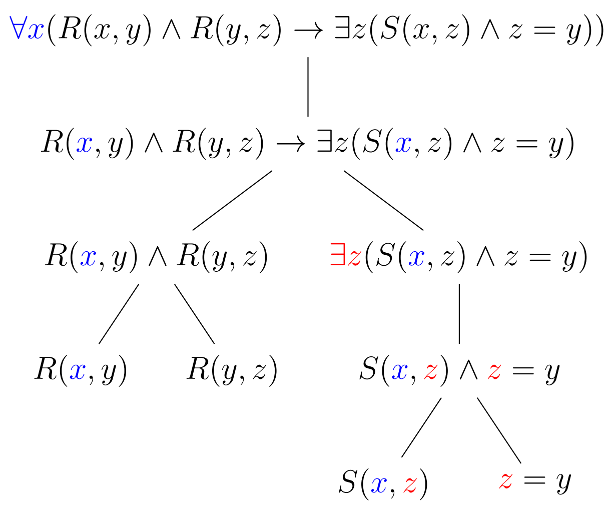
Truth in a Structure (Formal)

Once again, natural language is only a rough guide, and as engineers we need a more rigorous system. While we are able to work out the truth value of a complex formula by evaluating propositional atoms from the root of a formation tree in propositional logic, it's not as simple in predicate logic. For example, if we tried to evaluate $\forall x(\texttt{ate}(\texttt{Charlie}, x) \to \texttt{fruit}(x))$ with a formation tree, we'd quickly run into trouble;


$\forall x(\texttt{ate}(\texttt{Charlie}, x) \to \texttt{fruit}(x))$
$\texttt{ate}(\texttt{Charlie}, x) \to \texttt{fruit}(x)$
$\texttt{ate}(\texttt{Charlie}, x)$
$\texttt{fruit}(x)$

What are the truth values for the leaf nodes? We don't know, since formulas of predicate logic doesn't have to be true or false in a given structure.

With a given formula A, a variable x in an atomic subformula of A is **bound** if it's under a quantifiers in the formation tree. Otherwise, the variable is **free**. For example (copied directly from *First-order logic.pdf*);

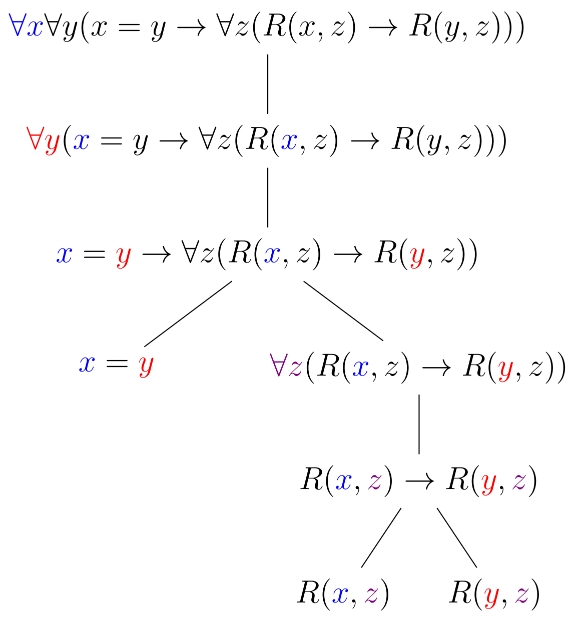

$\forall x(R(x, y) \land R(y, z) \to \exists z(S(x, z) \land z = y))$
$R(x, y) \land R(y, z) \to \exists z(S(x, z) \land z = y)$
$R(x, y) \land R(y, z)$
$\exists z(S(x, z) \land z = y)$
$R(x, y)$
$R(y, z)$
$S(x, z) \land z = y$
$S(x, z)$
$z = y$

The coloured variables are bound, and the uncoloured ones are free. Notice that z occurs as both a free, and as an unbound variable. The two instances of z are different, and have nothing to do with each other.

A **sentence** is defined as a formula without any free variables (hence all variables are bound).

For example, $\forall x(\texttt{ate}(\texttt{Charlie}, x) \to \texttt{fruit}(x))$ is a valid sentence, however the subformulas aren't $(\texttt{ate}(\texttt{Charlie}, x) \to \texttt{fruit}(x))$ isn't a sentence, since the x is free.

For example, take the slightly more complex formula $\forall x \forall y(x = y \to \forall z(R(x, z) \to R(y, z)))$; we can say it's a sentence. This can be proven by the following formation tree;


$\forall x \forall y(x = y \to \forall z(R(x, z) \to R(y, z)))$
$\forall y(x = y \to \forall z(R(x, z) \to R(y, z)))$
$x = y \to \forall z(R(x, z) \to R(y, z))$
$x = y$
$\forall z(R(x, z) \to R(y, z))$
$R(x, z) \to R(y, z)$
$R(x, z)$
$R(y, z)$

Evidently, there are no free variables, as all the atomic L-formula consist of bound variables.

Problems with Free Variables

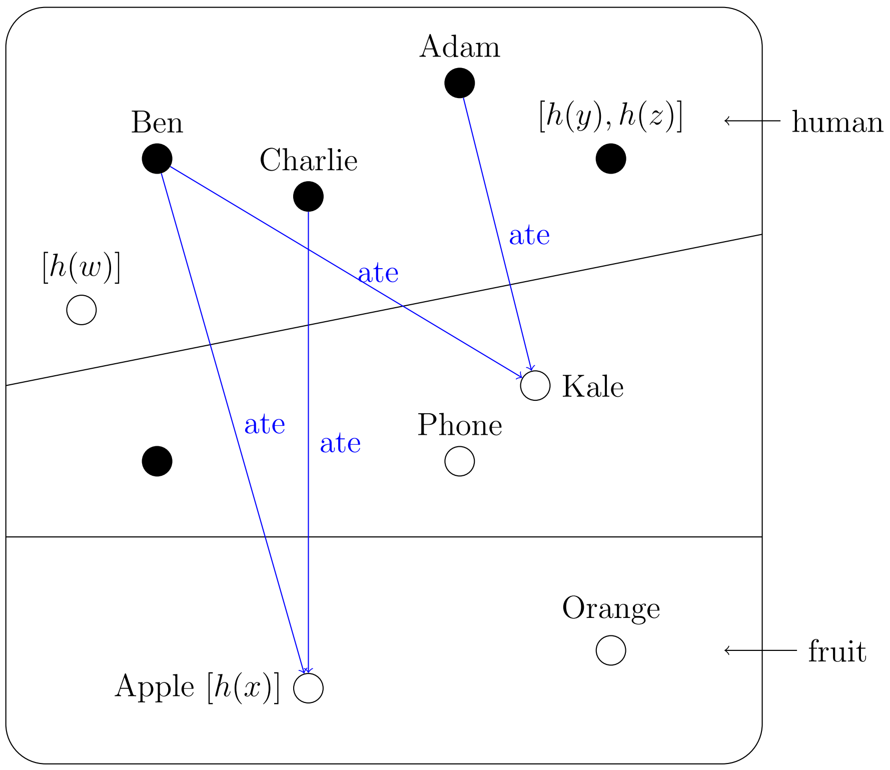
While a sentence can be evaluated to true or false in a structure, the same cannot be said for non-sentences. A formula with free variables doesn't evaluate to a truth value, seeing as they have no meaning in a given L-structure M. For example, $x = 7$ might be true, but we have no way of saying whether it is, since we don't know the value of x in M. Therefore the structure is an **incomplete** situation, since it doesn't fix the meanings of free variables. Note that we need to specify values for fre variables, even if they don't change the answer e.g. $x = x$.

We solve this with **assignments**; suppling a missing value to a free variable. **An assignment does for a variable the same as what a structure / model does for a constant.**

NOTATION: let there be a signature L, have an L-structure M, and let h be an assignment into M. Then for any given L-term t, the value of t in M under h is allocated by;

- t is constant M: t^M
- t is variable h: $h(t)$

Reusing the previously drawn model (M, h), because doing diagrams with TikZ is painful;



- the value of the term `Charlie` in M under h is the black node marked 'Charlie' $\texttt{Charlie}^M$
- the value of the term x in M under h is the white node marked 'Apple' $h(x)$