The Syntax of Propositional Logic

English (or, any *natural language*) is *ambiguous* and, as important, loaded with assumptions about its meaning. Missing (*elided*) words are inserted by the reader, and we are uncomfortable with certain readings.

- John drove on and hit a pedestrian.
- John hit a pedestrian and drove on.
- John is working or he is at home.
- John is working or the Pope Catholic.
- Euclid was Greek or he was a mathematician.
- John was not not a mathematician.
- John was not not not not a mathematician.
- If I open a window then we'll have fresh air.
- If the Pope is Jewish then Hodas is Irish.
- If the Pope is Jewish then we'll have fresh air.

The Syntax of Propositional Logic

We will fix a new and unambiguous notation:

Definitions: Fix some countably infinite set of *proposition symbols* (or *propositional letters*, or *propositional variables*), $P = \{p_0, p_1, p_2, \ldots\}$. The *propositional language over* P has as its alphabet the set

$$\Sigma = P \cup Conn \cup Const \cup Aux$$

where:

1. $Conn = \{\wedge, \vee, \Rightarrow, \Leftarrow, \neg, \equiv\}$ is the set of *logical connectives*.
2. $Const = \{\top, \bot\}$ is the set of *logical constants*.
3. $Aux = \{), (\}$ is the set of *auxilliary symbols*.

The Syntax of Propositional Logic

Definition: Given P and its extension Σ as defined above, the propositional language over P (abbreviated $PROP$), also called the set of *well-formed-formulas* (or *WFF*'s) of the language, is the smallest subset of Σ^* such that:

1. $\top \in PROP$, and $\bot \in PROP$.
2. For all $p_i \in P$, $p_i \in PROP$.
3. If $\Phi \in PROP$ then $(\neg\Phi) \in PROP$.
4. If $\Phi, \Psi \in PROP$, and $\bullet \in \{\wedge, \vee, \Rightarrow, \Leftarrow, \equiv\}$, then $(\Phi \bullet \Psi) \in PROP$

Cases (1) and (2) determine the set of *atomic formulas* of the language. We will denote this as the set $Atom$.

For all $p_i \in P$, the formulas $p_i, (\neg p_i) \in PROP$, are called *literals*. We will denote this as the set Lit.

Freely Generated Sets

The purpose of requiring the use of parentheses in the construction of the formulas is so that the set of all WFFs is what is known as *freely generated*. Roughly what this means is that there is a unique, identifiable way in which a larger element of the set is constructed from one or more smaller elements.

That is, each formula in $PROP$ has a unique *formation tree* (or *parse tree*). For example, the formation tree of the formula $((p_1 \Rightarrow p_2) \Rightarrow (p_1 \wedge p_2))$ is:

If we did not use parentheses, the set of formulas would not be freely generated and we could conceivably form more than one formation tree. For example, the formula $p_1 \Rightarrow p_2 \Rightarrow p_1 \wedge p_2$ would have several possible formation trees including:

Freely Generated Sets

The reason we care about free generation is that when a set is freely generated functions defined using cases that correspond to the formation rules for the set are guaranteed to be well defined.

Suppose we define a function f which gives numerical values to formulas according to the following rules:

1. $f(\bot) = f(\top) = 0$
2. For all $p_i \in P$, $f(p_i) = 1$.
3. If $\Phi \in PROP$ then $f(\neg\Phi) = -f(\Phi)$.
4. If $\Phi, \Psi \in PROP$, then:
   (a) $f(\Phi \Rightarrow \Psi) = f(\Phi) - f(\Psi)$
   (b) $f(\Phi \bullet \Psi) = f(\Phi) + f(\Psi)$ for all other connectives

$f(a \Rightarrow b \wedge c)$ then has two possible values, while $f(a \Rightarrow b \wedge c)$ has only one.

Freely Generated Sets

Formally, because the set $PROP$ is freely generated, we have the following theorem:

Theorem: For any set A and function $f : Atom \to A$, given functions $h_\neg : A \to A$ and $h_\bullet : A \times A \to A$ (for each $\bullet \in \{\wedge, \vee, \Rightarrow, \Leftarrow, \equiv\}$), there is a unique, well-defined function f' (called *the unique homomorphic extension of* f) such that:

1. $f'(\ at\ ) = f(at)$, for all $a \in Atom$
2. $f'(\ (\neg\Phi)\ ) = h_\neg(f'(\ \Phi\ ))$, for all $\Phi \in PROP$
3. $f'(\ (\Phi \bullet \Psi)\ ) = h_\bullet(f'(\ \Phi\ ), f'(\ \Psi\ ))$, for all $\Phi, \Psi \in PROP$

Why do we care? Well, truth is just a function on formulas. We presume that our semantics is *compositional*, that is, that the truth of a formula is dependent on truth of its *subformulas* in a well-defined way determined by the pattern of connectives.

Subformulas

Definition: Given two formulas, $\Phi, \Psi \in PROP$, we say that Φ *is a subformula of* Ψ if the formation tree of Φ is a subtree of the formation tree of Ψ. Φ *is a proper subformula of* Ψ if it is a subformula of Ψ and it is not the same formula as Ψ.

Note that if we use a visually ambiguous syntax for formulas and rely on precedence rules to determine the formation tree, then subformula is not equivalent to substring. For example, it may appear that $q \equiv \neg p$ is a subformula of $p \Rightarrow q \equiv \neg p \Rightarrow \neg q$, because it is a substring, but it is not.

If we use the full, freely-generated syntax with parentheses, we see that $(q \equiv (\neg p))$ is not a subformula of $((p \Rightarrow q) \equiv ((\neg p) \Rightarrow (\neg q)))$. In this setting substring and subformula correspond (provided the substrings are valid formulas).

The Semantics of Propositional Logic

The formulas $\top$ and $\bot$ will always have the meanings *true* and *false*.

The propositional letters are used to stand for atomic assertions about the world, such as "Hodas is a Professor", or "TG-101 is a rat hole", or "Logic meets on Tuesday". These may either be true or false depending on the world we are modelling. We assume however, that all propositions are one or the other.

Definition: A function $v : P \to \{true, false\}$ assigning *truth values* to propositional letters is called a *valuation*, or an *interpretation*, or a *truth assignment*.

The Semantics of Propositional Logic

The meaning of each connective is a function mapping truth values to truth values. These functions are generally presented in the form of *truth tables*.

$h_\neg =$

A	$(\neg$ A$)$
true	*false*
false	*true*

$h_\wedge =$

A	B	(A $\wedge$ B)
true	*true*	*true*
true	*false*	*false*
false	*true*	*false*
false	*false*	*false*

$h_\vee =$

A	B	(A $\vee$ B)
true	*true*	*true*
true	*false*	*true*
false	*true*	*true*
false	*false*	*false*

$h_\Rightarrow =$

A	B	(A $\Rightarrow$ B)
true	*true*	*true*
true	*false*	*false*
false	*true*	*true*
false	*false*	*true*

$h_\Leftarrow =$

A	B	(A $\Leftarrow$ B)
true	*true*	*true*
true	*false*	*true*
false	*true*	*false*
false	*false*	*true*

$h_\equiv =$

A	B	(A $\equiv$ B)
true	*true*	*true*
true	*false*	*false*
false	*true*	*false*
false	*false*	*true*

The Semantics of Propositional Logic

So, given the truth-table definitions of the functions providing meaning for the connectives, and given a valuation function v assigning meaning to each propositional letter, there is a unique (i.e. well-defined) function $v' : PROP \to \{true, false\}$ assigning meaning to all propositional formulas.

Remember that this is entirely dependent on $PROP$ being freely generated. If we had not used parentheses in the construction of strings in $PROP$, then the set would not be freely generated, and many formulas would have more than one possible formation tree. In turn, functions definied recursively on the structure of formulas would not be well-defined.

For instance, given a valuation assigning *false* to both p and q, the formula $p \Rightarrow q \Rightarrow p$ could have either truth value, depending on which instance of $\Rightarrow$ we assumed to be at the root of the formation tree.