Knowledge and Reasoning: Propositional Logic

We briefly mentioned much earlier that an agent receives percepts from its environment, and performs actions on that environment; and that the action sequence can be based on a plan that is formed by searching (blindly or in an informed way) among all possible such sequences. However, something much richer can go on in this plan-formation stage. The agent can learn (thereby becoming aware of new actions or consequences of actions), and it can reason with what it knows (possibly seeing quickly how to achieve something that might take much longer if done by ordinary searching).

Thus, an agent benefits a lot from having a knowledge base (KB – the explicit "factual" part of a KBN) that it can add to and draw on. And to draw on the KB means to access certain items in the KB and use them to guide its decisions, by some sort of reasoning process. That is, the agent will also have an inference engine at its disposal.

[Note: calling this a knowledge base should not be taken to mean that everything in it is unquestionably true. But it is to be thought of consisting of items that the agent takes to be true, at least for the moment. What is it to take something to be true? Here we get into difficult issues a bit far afield from our present concerns, but undoubtedly important for full-fledged human-level cognitive agents. In fact, even characterizing what counts as knowledge is very tricky.; if time permits we will look at this briefly later on.]

All this suggests that the agent will be using some form of logic, where the items in the KB are taken as axioms, and the inference process if governed by the logic's rules of inference. This was the view long championed by John McCarthy, in the so-called logicist approach to AI. And it still is one vigorous approach being pursued today, among others.

We will need to take a look at Propositional Logic and then First-Order Logic, before we can assemble the machinery to solve the Monkey & Bananas problem. Roughly speaking, the monkey will know (have in its KB) basic information about its environment (positions of box, bananas, and itself; actions it can take; etc) and what its goal is (have the bananas); and it will have access to inference rules to allow it to draw conclusions about what to do. The details are rather involved, but the basic idea is fairly simple.

However, a worry might arise: there are in general many ways to start proofs, and most of them won't lead to what we might want to prove. Aren't we then right back where we started, with a horrible blind-search problem? Won't we need a special set of search techniques particular to what we want to prove?

Well, it turns out that there is a clever way to approach this quite generally, and one that we already know: proof by contradiction. Instead of trying to prove a result R directly, we assume –R and try to prove a contradiction; if we succeed, then we have shown –R cannot be correct, hence we have shown R. And so all we need is one (powerful) technique for proving contradictions. We will return to this later.

PROPOSITIONAL LOGIC

A logic in general consists of

1. a formal language
2. inference rules
3. axioms
4. semantics (what the symbols mean)

Propositional logic (PL, for short) has a language consisting of letters (A, B, C, etc; so-called sentential variables), connectives (→, ^, v, ~), and parentheses; and the usual ways of combining these into so-called well-formed formulas (or wffs). At times we will want to refer to arbitrary wffs, and we can use Greek or boldface letters (A, B, etc) to indicate unspecified wffs that may be simple letters or more complex wffs such as A&~C, C→(BvC), etc.

The inference rules for PL can be of various sorts. A very common choice is simply that of modus ponens (MP): from A → B and A, infer B. It is often enough all by itself; what we mean by enough will become clear later.

As for axioms, a simple choice is to allow all tautologies: wffs that are always true – but to say what that means will require us to loo at semantics, so we turn to that now.

Semantics for PL is that of the familiar truth tables. We simply list all the letters of interest, and under them we create rows with all possible arrangements of Ts and Fs. Each such row is called an *interpretation* of PL. It does not really go so far as to state meanings for the letters; it simply marks each as true or false. And then we extend the listing to include whatever wffs we are interested in (that employ those letters), and we also mark down the proper Ts and Fs for them in each row, based on the usual understandings. For instance, the truth table for A → B is

A	B	A→ B
T	T	T
T	F	F
F	T	T
F	F	T

I find it easy to remember this as follows: consider → to indicate a promise: if you do what is on the left-hand side, then I promise to do what is on the right-hand side. Now, how can it happen that the promise A→ B turns out to be false (that is, how can I break the promise)? You will have to do your part (A) and I will have to refrain from doing my part (B). That is, A→ B is false iff A is true and B is false, which happens only in row 2. So that row gets marked F and the other three T.

Now we can say precisely what a tautology is: any wff whose truth-table assigns it the value True in every row. And a contradiction is a wff that is assigned False in every row. Finally, a contingent wff is one that is neither a tautology or a contradiction: it is assigned both True and False (in different rows, of course).

So, the tautologies are pretty special. But they can take a great many forms, and there is an infinite number of them. So it is a bit awkward taking all of them as axioms, when we don't even know what they all look like. A more streamlined set was proposed by Lukasiewicz:

1. **(~A → A) → A**
2. **A → (~A → B)**
3. **(A→B) → ((B→C)→(A→C))**

The three forms above are schemata: they each represent an infinite number of possible wffs, gotten by replacing the boldface letters by any actual wffs we like. Each of the three forms however always will be a tautology (this is easily verified by truth-tables), so the Lukasiewicz schemata (let's call them L) are a subset of all tautologies.

It turns out that the rule MP together with axioms L lead to exactly the same set of consequences (things that can be proven) as does MP with all tautologies as axioms. However, it is still cumbersome that L represents an infinite number of axioms. We will see a way around that a bit later on.

What does it mean to prove something, say the wff W, in PL? It means: to start with whatever one is using as axioms (which need not consist of tautologies, by the way – just whatever one wants to take to be true for whatever reason), and to apply one's rule(s) of inference over and over until one arrives at the result that is wanted. If this can be done, we write

Ax |– W

where Ax is our chosen set of axioms (we perhaps could call it KB instead); the inference rules are understood to be whatever we have specified. This notion of proof is formal, or syntactic: it does not make any mention of truth or meaning. We sometimes read the above as "Ax proves W", which really means: there is a proof of W from Ax (using the understood rules of inference, eg MP).

There is another totally different way to try to “conclude” a wff: write out the truth-table for all the axiom wffs as well as the wff W that one wants to prove. Then check all those rows in which the axioms are all true, to see if W holds in them. If so, we say W is *entailed* by the axiom set (Ax, say), and we write

Ax |= W

This means, then, that W is true in all interpretations in which all wffs of Ax are true. It is a semantic notion, making use of truth-values, but does not use inference rules at all.

So we have two extremely different notions of “arriving at” a wff as a conclusion, starring from initial (axiom) wffs. One way is much like the usual mathematical notion of proof, where we reason step-by-step, starting from axioms and using inference rules as we go along, trying to be clever enough to choose the right axioms and the right rule(s) to get to the result we want; and then it looks much like a search problem, in a huge tree of possible proofs-sequences.

And the other method is much more routine, possibly tedious, but very methodical: just fill out the (possibly enormous) truth-table and check to see of W is true in every row where all the axioms are true. There is no thinking needed here, no cleverness.

How do they match up? Well, with the right combination of axioms and inference rules, they match up perfectly. For instance, we have this famous result:

Theorem (Soundness and Completeness of PL): Consider any version of PL that includes the Lukasiewicz schemata among its axiom set Ax, and MP as one of its inference rules. Then for every wff W, Ax |= W if and only if Ax |– W.

The left-to-right direction is completeness (the proof method is able to prove everything it should), and the right-to-left is soundness (everything provable from Ax is true whenever Ax is).

A digression on speed/efficiency: Even with the Lukasiewicz axioms – indeed even with the much more streamlined Robinson resolution method that we will see shortly – it still usually takes a long time to find proofs in PL. There is a famous problem, called SAT: the satisfiability problem for PL. It is this: given a wff W, is it satisfiable (is it true in at least one interpretation) or not? We can settle the matter by drawing the entire truth table for W, of course, but that is very time-consuming, requiring in general the filling out of as many as 2^n rows, where n is the number of letters in W, and where each row has at least n+1 entries (one for each letter plus one for W). We say that this solves SAT, but it is an exponential time and space solution.

SAT is in the class of problems known as NP-problems. This is the class of problems that, when a supposed solution is suggested, it is easy (i.e., can be done in time proportional to a polynomial in the "size" n of the problem) to check if it really is a solution. In fact, SAT was the first problem shown to be NP-hard: any solution to SAT can be transformed into one that solves any other NP problem in essentially the same amount of time. So SAT is maximally hard among NP problems. No one has ever found a fast (polynomial-time) solution to SAT; and if anyone does, then that will mean all NP problems can be solved in polynomial time.

What makes SAT so special like this? Well, the idea is that, being based on such a basic logic framework, it can be used to encode all the other NP problems. This idea was exploited by Stephen Cook in 1971, when he first proved that SAT is both NP-hard and a member of NP (together these say that SAT is NP-complete).

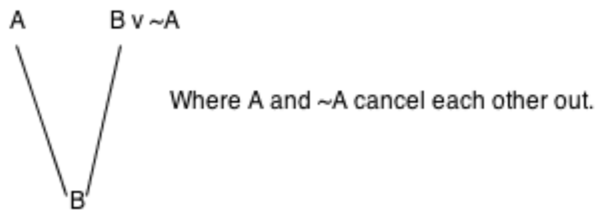
One final note on PL proofs: to prove W from a given axioms set, KB, we can suppose ~W (enlarge our axiom set to KB' = KB + ~W) and try to show that KB' is inconsistent (contradictory). This we mentioned before: proof by contradiction, also called indirect proof. In particular it is typically used along with Robinson's resolution method, and also is basic to the PROLOG programming language.

RESOLUTION in PL

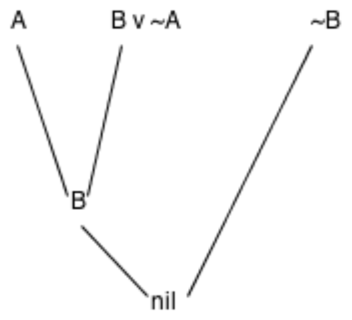
Here we take an initial look at Alan Robinson's **resolution method** of inference.

To prove **B** from **A** and **A → B**, we can use modus ponens.

But there is another way, in which we first rewrite the if-then wff as **B v ~A**. (Question: what is meant by "rewrite as"?) Then we proceed as indicated in this diagram:



Or here is another variation of the above: we add **~B** as another "axiom" and then proceed:



Here ~B is introduced to allow a proof of B by contradiction, aka an indirect or **refutation** proof.

The conclusion *nil* means that the assumption **~B** must be rejected, hence **B** has been proven to follow from **A** and **B v ~A**.

This is a simple example of the **resolution-refutation** method introduced by Robinson in 1963. We have shown an example in PL (but it has an extension to FOL as well). *Robinson showed that it gives the same results (the same things can be proven) as in regular PL with MP and the Lukasiewicz tautologies.* Notice that *no* axiom-tautologies are needed at all! And the only inference rule is that of canceling terms such as **A** and **~A**. There is some extra work needed in preparing the wffs, by rewriting some of them in equivalent form, for instance to eliminate → in terms of **v** and ~.

Definition: A wff is in **conjunctive normal form** (CNF) if it has the form of a conjunction of disjunctions of literals. (A conjunction consists of **&**-separated expressions, and a disjunction of **v**-separated expressions. A literal is either a sentential variable/letter or its negation.)

Thus the following are in conjunctive normal form:

A, A v ~B, A & (A v ~B), B&A, (BvA)&(~BvC)&(Cv~A). [Note these are not bold schema-letters; they are actual PL letters.]

And these are not in CNF: A → B, ~(AvB).

A disjunction of literals is called a **clause**. Thus a wff in CNF is a conjunction of clauses.

It turns out that *every* PL wff is equivalent to a wff in CNF. (We say two wffs are *equivalent* if they have the same truth-table values in all interpretations. This answers the above question about rewriting a wff. This is <u>*not*</u> the same as saying they are *equal*; two different wffs are never equal.)

[If we want to give examples of arbitrary CNF wffs, we can use letter-schema-variables such as **J, K, L** which are understood to refer to individual (but unknown) PL letters (not arbitrary wffs as in the case of **A, B, C**, etc). So this example is in CNF form even though it is not an actual wff since we don't know which letters are really involved: **L & (~L v K) & (L v ~K).** All we know is that **L, K** each refers to *some* letter such as A or B or P or Q.]

To find a CNF wff equivalent to a given wff W, one uses well-known equivalences such as the ones below.

deMorgan:

~(**A**v**B**) is equivalent to ~**A** ^ ~**B**
~(**A**^**B**) is equivalent to ~**A** v ~**B**

distributivity of ^ over v:

(**A**^**B**) v **C** is equivalent to (**A** v **C**) ^ (**B** v **C**)

implication-elimination:

A → B is equivalent to ~**A** v **B**

In addition, if a literal appears more than once in a clause, removal of one of them produces an equivalent result. Finally, if a literal and its negation both appear in a clause, then the entire clause is a tautology and hence equivalent to any other tautology.

The general form of a resolution step for propositional logic involves two clauses, one including the literal **L** and the other **~L**, and the result is to infer a new clause that is a disjunction of all the *other* literals in both clauses. Thus from **Lv~KvJ** and **~Kv~LvM** one may infer **~KvJv~KvM** which is equivalent to **~KvJvM**.

We will return to resolution-refutation after we have reviewed FOL (first-order logic).

But first, one more example, further illustrating the power of ref-res (refutation-resolution):

Without special axioms, how can we prove a tautology? For instance, how can we prove ~(P^~P)? Simple. We just follow the steps. WE assume the negation of what is to be proven, and use that to derive a contradiction. The negation in this case is:

~(~(P^~P))

and we must find an equivalent wff in CNF. But our wff is equivalent to
P^~P, which we then separate into the two clauses P, and ~P. These then resolve to yield the box, i.e., a contradiction. And we are done.