Things, Classes, and First-Order Logic

Propositional logic (PL) is very useful. But it falls short for our needs. A reasoning agent will need to be able to refer to specific entities and events in the world, and to process assertions about them. In a sense, it parallels the "plight" of machine learning. While it can do a lot, it cannot take account of particular things in the world, nor relate them to general properties.

Nevertheless, PL – and modus ponens in particular – point in a promising direction; they *seem* to involve a notion of classes of things (or situations). Thus if we use P to mean "it is raining" and Q for "the grass is wet", then $P\rightarrow Q$ seems at least plausible, and we may think we then have at hand notations for grass and wetness, or at least for cases where it is raining, etc. But this is an illusion: PL has no symbols for things or their properties, or even for cases, rather just for an entire sentence at once, each one a sort of black box whose details are invisible. When we assert P as well as $P\rightarrow Q$, we mislead ourselves if we think we have written "imagine a situation where P is true" and "in any situation where P is true, so is Q". To be sure, a truth-table row (interpretation) where P and $P\rightarrow Q$ are both marked T is taken as a kind of substitute for such a situation. But there can be many such situations corresponding to just one row, and neither the syntax nor the semantics of PL distinguishes them at all (rain today, rain yesterday), let alone their details such as rain and grass, etc. One might provide a separate letter for each individual case: P for rain today, Y for rain yesterday, and so on; but (i) that would be a mess, and (ii) it still would not provide a way to refer to rainy days in general, nor to note that it rained two days in a row.

What we need are symbols that can refer to things, events, properties, and so on. This is what first-order logic (FOL) provides, along with the capacity to refer to collections of things (via quantified variables). FOL is – arguably – just what we need for representing sophisticated real-world reasoning. It will turn out that we can get very far indeed with FOL, although tweaks (some large, some small) will be needed at various stages.

Let us briefly comment again on the notion of classes of things. ML is often used to classify data, and sometimes (in unsupervised learning) to come up with classes on it own. Thus the know-how (e.g., how to classify kinds of fruit) can be thought of as providing the basis for the notion of a particular class (e.g., the class of bananas). "All" we would need is a process that takes the learned know-how and produces "tags" for the classes. Thus again, the KBN seems to have two parts: the KN (knowledge-network) part, and the KB (knowledge base) part. The connection, while obviously important, is highly non-trivial, however, and still the subject of ongoing investigation. We'd like an agent to conclude "this is a banana"; but while we can get an output unit to fire when a banana image is input, we violate McDermott's warning when we think the job is done. The English sentence holds a great deal that is nowhere to be found in the output: "a banana" versus bananas; "this" vs some or any; banana vs mere output-unit firing, and so on. Know-how and know-that are different in many ways. And know-that itself is very complex, as we will see later on.

Review of FOL

1. **Language:**
    - the same connectives as for PL, parentheses, and a comma
    - variable symbols x,y,z, etc
    - constant symbols a,b,c, etc
    - predicate symbols P, Q, R, etc
    - function symbols f,g,h, etc
    - quantifiers ForAll and Exists
    - wffs are formed according to the familiar rules, where each predicate symbol takes the appropriate number of arguments

    Examples:

P(c)
Exists y (Q(y,b))
ForAll x (P(x) → ~Q(x,f(b)))
Q(a,b) ^ ForAll y (Exists z (P(g(y)) v –Q(z,y)))

2. **Inference rules**
    These usually include a generalized version of MP: from ForAll x [Px → Qx] and Pc, infer Qc; and also some additional rules such as: from P(a) infer Exists x P(x), and from ForAll x P(x) infer P(a).

3. **Axioms**
    Here there are special sets of wffs that play a role like that of the Lukasiewicz axioms in PL. But when we come to FOL resolution we will not need these (nor will we need the MP rule).

4. **Semantics**
    Here we have a great deal beyond what PL has to offer. An interpretation for FOL actually assigns entities to the constant symbols, meanings to the predicate symbols, functions to the function symbols. We don't simply *say* a wff is true or false, as in a truth-table; we give it so much meaning that it really is true or false in a given interpretation. To do this, we first specify the *domain* D, consisting of all the objects that we wish the constants and variables to refer to. Then we specify actual specific members of D to be what the constant symbols refer to, actual functions for the function symbols, and actual relations on D for the predicate symbols.

Example: Let interpretation I_1 be as follows: D = N (the set of natural numbers: 0,1,2,…), let c stand for 0, and d for 1, f for the successor (add 1) function, = for equality, and G for greater than.

Then the wff

G(f(c),c) & =(f(c),d))

is true in I_1. We'd more normally write it as

(f(c) > c) & (f(c)=d)

and it means (in I_1)

(0+1 > 0) & (0+1 = 1),

which is indeed true.

But now consider interpretation I_2: D = {all living people }, c means Judy, d means Tom (Judy's father), f means father-of, = still means equality, and G means older than. Now the very same wff

G(f(c),c) & =(f(c),d))

means in interpretation I_2 that Judy's father is older than Judy, and Judy's father is Tom, which is still true.

Finally, if we let I_3 be the interpretation that is just like I_2 except that now the meaning of G is younger-than, then the same wff now means Judy's father is younger than Judy, and Judy's father is Tom, which is false (the conjunction is false, even though part of it is true).

We will suppose someone has given us a particular set of wffs to use as axioms, which we will refer to collectively as the KB (knowledge base). (Later on it will turn out that this needs to be modified, but we can go a long way with the current notion.)

As before, we use KB |- **W** to indicate provability of **W** from KB, and KB |= **W** to indicate truth of **W** in all interpretations in which the wffs in KB are true. We call such an interpretation a *model* of the axiom set KB. We also sometimes refer to any set of axioms that we wish to consider, a *theory*.

In FOL, we still have a **soundness and completeness theorem** as in PL: Given any axiom set KB, then for every wff **W**,

KB |= **W** if and only iff KB |- **W**,

where the proof methods are suitable ones to make this work (for instance, refutation resolution). Another way to state soundness and completeness is that a wff is provable in a theory (KB |- **W**) if and only if it is true in all models of the theory (KB |= **W**). Thus in FOL the notion of a model takes the place of that in PL of a truth-table row where all the antecedent wffs are true.

A glimpse ahead

We want our agent to be able to reason, i.e., to augment its KB by proving new wffs from ones it already has. Partly this is good in itself: knowledge is good to have. But also, if the agent is going to *do* anything, it will need to make plans and decisions; this will come later, when we study how to *use* the KBN. A key example is that of the famous Monkey-and-Bananas Problem: how can a monkey eat bananas that it sees hanging just out of reach, in a room where there is a sturdy box in one corner?

We are very slowly closing in on being able to write down the KB for the hungry monkey, but not quite. The monkey's world changes as it performs actions. So we need a way to indicate different states of the world. (This is true in any planning domain, or even any search problem.) Thus we need our domain D to be one that includes not only the monkey, the bananas, and the box, but also situations the world can be in. For instance, locations can change; so we need constants to indicate locations as well as entities to indicate when the box is at one location and when at another, etc.

So we will eventually introduce a situation variable, s. This is not an addition to FOL. It just means s will be one of the variable letters, and we keep in mind what we intend by it. We also will need a start-state (or start-situation), which will be a constant, say s_0. Finally, we'll need function symbols for the possible actions: push, climb, grasp. There are other details we will need as well. But all that will come in a later section, when we turn from learning knowledge to using knowledge.

Next we will turn to the resolution refutation method in FOL.

FOL RESOLUTION AND UNIFICATION

As in PL, resolution involves first rewriting the axioms (KB) in CNF. But now it is more involved.

1. Eliminate implications (→)
2. Move ~ inwards
3. Standardize variables to avoid clashes
4. Skolemize (replace existential quantifications by special functional expressions)
5. Drop universal quantifiers
6. Distribute v over &.

We will eventually do an example, modified from one in the Russell-Norvig book:

ForAll x [(Hx & ForAll y (Ay → Lxy)) → Exists y (Hy & Lyx)]

One interpretation is that every person who loves all animals is loved by some person. Here H is interpreted to indicate humans, A animals, and Lxy means x loves y. The intended domain would include all people and all animals.

We will consider this example in detail momentarily. But first we give a few “warmup” examples:

1. Suppose we want to prove Qc from Pc and ForAll x, Px → Qx, along lines similar to what we did in PL resolution. We first need to make sure all the wffs are in (equivalent) CNF form. Qx and Px already are. But what about the universally quantified wff? (FOL, CNF means a conjunction of disjunctions of individual predicate letters (with arguments) or their negations.)

We rewrite the if-then part as usual, get ForAll x (Qx v –Px). Next we drop the ForAll x, with the understanding that any variables will be regarded as universal (how to deal with existential quantifiers will come soon). So now we proceed as follows:

$$\begin{tikzcd}
Pc \arrow[dr, dash] & Qx \text{ v } {-Px} \arrow[d, dash, "x \leftarrow c"] & \underline{-Qc} \text{ (the negation of what we want to prove)} \arrow[dl, dash] \\
 & Qc \arrow[d, dash] & \\
 & \text{nil} & \text{And so we have proven Qc.}
\end{tikzcd}$$

What has happened here? The (universal) variable x has been replaced by the constant symbol c. This is an example of *unification*, a crucial part of FOL resolution.

In general, two non-predicate expressions (made of variable and/or constants, and possibly function symbols) “unify” if one can make substitutions for the variables in such a way that the two become literally identical. And in that case, one goes for the *most general unifier* that does the trick. Thus for instance:

x and f(x) cannot unify; nor can f(x) and g(x), nor c and f(x), etc

f(x) and y unify by substituting f(x) for y; also by substituting, for instance, c for x and f(c) for y; or g(b) for x and f(g(b)) for y, etc. But of these, the first choice is the most general: it makes the fewest commitments and leaves open the most possibilities for further unifications later on.

2. Suppose we are given P(g(c)) and want to prove Exists x Px, where g is a function symbol. P(g(c)) is already in CNF. But we need to negate what we want to prove: – Exists x Px. This is equivalent to ForAll x –Px (sometimes this is even taken as the definition of Exists). Then proceeding as above:

$$\begin{tikzcd}
P(g(c)) \arrow[dr, dash] & & \underline{-Px} \arrow[dl, dash, "x \leftarrow g(c)"'] \\
& nil &
\end{tikzcd}$$

And so we have proven –Px, which really means ForAll x –Px, which is equivalent to Exists x Px. [Note: in FOL, wffs are equivalent if they have the same truth-value in all *interpretations*, much as in PL, except that an FOL interpretation is vastly more than a PL interpretation.]

3. Suppose P(c) and we want to prove Exists x P(g(x)). If we proceed as in example 2, we might find ourselves working with the wffs P(c) and -P(g(x)). And now we are stuck. There is no unifier for c and g(x)!

And this is good, because in fact, Exists x P(g(x)) is not entailed by P(c). How can we be sure of that? Well, we need to show that it is *not the case* that Exists x P(g(x)) holds in all models of P(c) (interpretations where P(c) holds). And one way to do that is to come up with an example of an interpretation where P(c) holds but Exists x P(g(x)) does not. So we can try to find a domain D and a function g* that g can represent, and an element c* of D that c can represent, so that c* can never be a value of g*. [It's easy, and there are many many ways to do it, with numbers, people, whatever.]

4. Suppose we know Exists x Px and also ForAll x (Px → Qx), and want to prove Exists x Qx. The universal wff is easy to convert [Qx v –Px]; what about the existential one? If we simply remove the Exists x in front, we'll have an unquantified variable (called a *free* variable) and it will be used as if it were universal as in example 1 above, not what we want (and not equivalent to the wff it came from). The x in this case really represents a single but unknown item, and we'd like to use a constant symbol in its place. But we dare not use any symbol already in the language since it might also be present in some other wff that we could use later on, and we cannot assume the two wffs are "about" the same thing. So, we introduce a *new* constant symbol into the language, called a *Skolem constant*; for instance s1, say. Then we write P(s1) as a replacement for Exists x Px.

But there is one thing we need to do before using s1. The variable x appears not only in Exists x Px, but also in the other axiom and in the "goal" wff that we want to prove. The goal will be negated and then replaced by ForAll x –Qx, and then by –Qx. This causes a clash with the x in Qx v –Px that we get from the other axiom; there is no reason that the two xs should refer to the same thing. So we need to "standardize" the variables, by making sure variables that are in the scope of different quantifiers are renamed so as to be different variable symbols. Thus we can, for instance, rewrite the universal wff axiom as ForAll y (Py → Qy) which then transforms into Qy v –Py.

So now we have P(s1) and Qx v –Px, and the negated “goal” –Exists x Qx, giving us the following:

```
  P(s1)                 Qy v –Py                  -Qx
      \  _____________  /                           /
               | y ← s1                            /
           Q(s1) ________________________/  x ← s1
                                   Nil
```

5. Now at last we return to the main example about animals and people and love, which we repeat here:

“Everybody who loves all animals is loved by someone.” This we represent in FOL as follows:

ForAll x [(Hx & ForAll y (Ay → Lxy)) → Exists y (Hy & Lyx)]

When we have finished looking at the example *per se*, we will then include another axiom, to the effect that “No one loves the person Charlie”, represented as Hc & -Exists x (Hx & Lxc), and from these two wffs we’ll prove Exists x (Ax & -Lcx).

We need to convert the main wff above to CNF:

1. eliminate implications
2. move negations in
3. standardize variables
4. Skolemize existentials
5. drop universals
6. distribute v over &.

In step i. we get
    ForAll x [-(Hx & ForAll y (-Ay v Lxy)) v Exists y (Hy & Lyx)]
Step ii. gives
    ForAll x [(-Hx v -ForAll y (-Ay v Lxy)) v Exists y (Hy & Lyx)]
and then
    ForAll x [(-Hx v Exists y -(-Ay v Lxy)) v Exists y (Hy & Lyx)]
then
    ForAll x [(-Hx v Exists y (--Ay & -Lxy)) v Exists y (Hy & Lyx)]
and finally
    ForAll x [(-Hx v Exists y (Ay & -Lxy)) v Exists y (Hy & Lyx)]

Step iii. replaces y in one Exists y(…) with a *new* variable, say z:
    ForAll x [(-Hx v Exists y (Ay & -Lxy)) v Exists z (Hz & Lzx)]

Step iv. replaces existentially quantified variables with Skolem functions (since y and z may depend on the universal choice of x:

ForAll x [(-Hx v (A(f1(x)) & -L(x,f1(x)))) v (H(f2(x)) & L(f2(x),x))]

Step v. gives

[-Hx v (A(f1(x)) & -L(x,f1(x)))] v [H(f2(x)) & L(f2(x),x)]

And step vi. gives

[(-Hx v A(f1(x))) & (-Hx v –L(x,f1(x)))] v [H(f2(x)) & L(f2(x),x)]

At this point, for ease of reading, we introduce abbreviations for the predicate expressions, in the same left-right order:

[(P v Q) & (P v W)] v [R & S]

then

{ R v [(P v Q) & (P v W)] } & { S v [(P v Q) & (P v W)] }

{ (R v (PvQ)) & (R v (PvW)) } & { (S v (PvQ)) & (S v (PvW)) }

(R v P v Q) & (R v P v W) & (S v P v Q) & (S v P v W)

and substituting back we get (whew!)

(H(f2(x)) v –Hx v A(f1(x)))
& (H(f2(x)) v –Hx v –L(x,f1(x)))
& (L(f2(x),x) v -Hx v A(f1(x)))
& (L(f2(x),x) v –Hx v –L(x,f1(x)))

CNF at last!!!!

Note the instances of x above are universal, so the clauses (the parts between the &s) are independently true for all x, and thus we can use different variables for each of them if we like.

Now let’s introduce the axiom Hc & -Exists x (Hx & Lxc) and the goal Exists x (Ax & -Lcx). These become (when the goal is negated)
Hc & (-Hy v –Lyc), and -Az v Lcz, where I have changed to new variables.

So we end up with seven clauses (disjunctions of literals):

1. H(f2(x)) v –Hx v A(f1(x))
2. H(f2(x)) v –Hx v –L(x,f1(x))
3. L(f2(x),x) v -Hx v A(f1(x))
4. L(f2(x),x) v –Hx v –L(x,f1(x))
5. Hc
6. -Hy v –Lyc
7. -Az v Lcz

Our aim is to derive a contradiction. We start by unifying 5 with each of 1-4, to get 1'-4', substituting c for x in all. These then are

1'. H(f2(c)) v A(f1(c))
2'. H(f2(c)) v –L(c,f1(c))
3'. L(f2(c),c) v A(f1(c))
4'. L(f2(c),c) v –L(c,f1(c))

Now we try to unify 1' and 7, since one contains A and the other –A. Using z ←ßf1(c) we have

8. H(f2(c)) v L(c,f1(c))

Unify 8 with 2':

9. H(f2(c))

From 9 and 6 with y ← f2(c):

10. –L(f2(c),c)

From 10 and 3':

11. A(f1(c))

Now we unify 4' and 6, with y ← f2(c)

12. –H(f2(c)) v –L(c,f1(c))

From 12 and 9:

13. –L(c,f1(c))

From 13 and 7 with z ← f1(c):

14. -A(f1(c))

From 14 and 11 we get nil.

This was *not* obvious, at all. There were very many choices that could have been made for wffs to unify; I have simply shown a fairly quick path leading to nil, out of the very very messy tree of possible paths, most of which lead nowhere interesting.

Fortunately, there are computer programs that can do the search for us. This is a part of the field known as automated theorem-proving. But it probably still will be a very long time before mathematicians are put out of business! Only very, very few new theorems have been first proven by automated means.

Yet if we adopt a simplified language, restricting wffs to so-called Horn clauses, then the search is often very fast indeed. A Horn clause is a disjunction of literals, where at most one is positive. The PROLOG language is a rather popular one of this sort. Alas, while faster, the restriction to Horn clauses significantly limits what one can write down. In particular, it is not adequate for the monkey problem.