ANSWER EXTRACTION

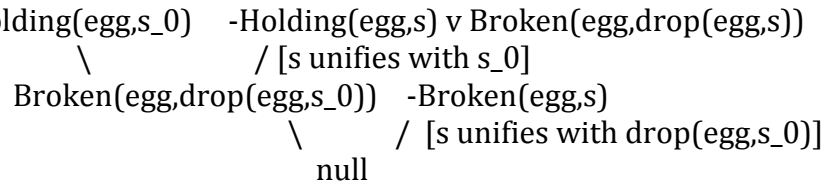
There is a trick that can be used in the resolution process, that finds the bindings (unifications) that are done to the hoped-to-exist (and sought-for) state s that transform it into a true goal state. For instance, in the Monkey-Bananas problem, we expect to show by resolution that (Exists s) Has(m,f,s). But succeeding at this does not produce an actual s for us. The resolution method works by using the negated goal in CNF form – in our case NG – that has this s in it, and the resolution leads to the null clause, indicating a contradiction. But there is no s left at that point! Along the way, s has become more and more refined via the use of actions, but these can come in in various orderings depending on how the resolution is carried out, and not all of them are guaranteed to lead to an actual plan. We will demonstate the trick by another much simpler example:

How to break an egg: drop it! Now let's do this in sit-calc:

INIT: Holding(egg,s_0)
Goal: (Exists s)Broken(egg,s)
NG: -Broken(egg,s)
Axiom: Holding(egg,s) -> Broken(egg,drop(egg,s))

CNFs:
Holding(egg,s_0)
-Broken(egg,s)
-Holding(egg,s) v Broken(egg,drop(egg,s))

Proof of null clause:



Here is the trick ("answer extraction") to get the plan: when the CNF of NG is used, adjoin to it (as a disjunct) the goal itself (actually the negated NG): NG **v G**, as in

-Broken(egg,s) **v Broken(egg,s)**

where **...** is used to tell us that we do NOT want to resolve that part away. (A perhaps safer – though less intuitive – notation for G might be ANSWER(s).) The purpose of the ***v G*** is to keep track of all the things the s in NG changes into via unifications during the proof. But since NG is not even there at the end (only nil or the "box" indicating a contradiction is left) we need something like **v G** to keep a record.

So here is the proof again, this time with answer extraction:

```
Holding(egg,s_0)     -Holding(egg,s) v Broken(egg,drop(egg,s))
          \                    / [s unifies with s_0]
    Broken(egg,drop(egg,s_0))    -Broken(egg,s) **v Broken(egg,s)**
                              \           /  [s unifies with drop(egg,s_0)]
                  {v Broken(egg,drop(egg,s_0))}
```

So now instead of getting null, we get the record of what s has become: a full description of the state that makes G true. And so the plan here is drop(egg,s_0). It is a good exercise to use answer extraction in the Monkey & Bananas problem to produce the plan automatically. Note that the example above brought in **v G** almost at the end; but it needs to come in wherever NG is brought in, early, late, or middle. (A good general rule is to try to bring NG into the proof as soon as possible.)

So, we now have seen how situation calculus can solve planning problems. But it is not perfect, due to the frame problem, and also because it needs a full-scale theorem-proving engine (which can be slow, and in general can encounter infinite-depth proof trees). And it shares -- with most other approaches to planning and reasoning -- difficulties in dealing sensibly with an inconsistent KB.

PROLOG is, by the way, NOT a full-scale FOL engine. It uses only so-called Horn clauses as axioms; these are clauses of the special form

P :- Q1,Q2,...,Qn [read: P if Q1 & Q2 & ... & Qn]

which is equivalent to -Q1 v -Q2 v ... v -Qn v P. (Here each letter stands for single non-negated predicate expression. Thus, *at most one* non-negated predicate expression may occur as a disjunct.) This means that even the very simple wff P v Q is not a Horn clause and cannot be used as an axiom in PROLOG. PROLOG tends to be very efficient, but this is in large part due to its restriction to Horn clauses. There has been much research in pushing this constraint aside in various ways. Note that the Monkey & Bananas problem, in the form we presented it above, has one axiom whose CNF is not a Horn clause (can you see which?).