An Induction Exercise concerning Evaluation

Thomas Forster

December 30, 2014

I'm assuming that you are happy with truth-table definitions of the operations $\wedge$ and $\vee$ on booleans (perhaps you prefer `AND` and `OR`). A **valuation** is a function from propositional letters to booleans. You can think of a valuation as a row of a truth-table, an assignment of boolean values to every propositional letter in sight. Given a complex formula A and a valuation v, one can *evaluate* A according to v and obtain a truth-value.

Clearly there is a function E: formulæ $\times$ valuations $\to$ booleans.

Exercise 1 *Write code for such a function E in your favourite functional programming language.*

(In case you were wondering, your favourite functional programming language is `ML`!)

The recursion you have written does not contain any instruction as to the mechanics of calculating the answer. One can evaluate *lazily* or *strictly*. For example: suppose that at some point in the computation-of-the-truth-values-of-A-according-to-v you have determined, for some subformula A' of A, that A' is `false` according to v. Suppose $A' \wedge B$ is the next subformula for us to tackle. If we are evaluating "lazily" we can get clever and say "Ah! Since A' has evaluated to `false` we know already that $A' \wedge B$ must evaluate to `false`, so we don't need to compute the truth-value of B!" In contrast, the strategy of *strict evaluation* requires us to compute the truth value of all subformulæ of an input before we try to compute the truth-value of the input. Thus there are **two** functions E_s and E_l (strict evaluation and lazy evaluation ...) but these two functions have *three* arguments not two. Their three arguments are: a formula, a valuation, and a *time*; and the recursive declaration for E_l will have base clauses like

If A is atomic, then $(\forall n \in \mathbb{N})(E_s(A, v, n) = E_l(A, v, n) =: v(A))$.
and

$$E_s(A, v, 0) = E_l(A, v, 0) =: v(A),$$

and recursive clauses like (for example)

$$E_l(A \vee B, v, t+1) = \texttt{if } E_l(A, v, t) = \texttt{ true then true else}\ldots$$

Exercise 2 *Write code for E_l and E_s in your favourite functional programming language.*

It's probably obvious to you that E_s and E_l will give the same end result. Perhaps it even looks *so* obvious that you think it's not actually worth proving. However, this fact *is* worth proving, and for two reasons. One is that it is a useful exercise in induction, and the other is that if our valuations are allowed to be partial functions it ceases to be true!

Exercise 3 *(Easy)*
Find A and v to illustrate how $(\forall t > 0)(E_s(A, v, t) \neq E_l(A, v, t))$ can happen if v is not total.

However, things are better behaved if we assume that all our valuations are total functions. So, let's make this assumption *pro tem*. Then, as we observed above, E_s and E_l will give the same end result. It's worth thinking about how one would state this last fact properly. So shield the rest of this pdf from your eyes for the moment and give the matter some thought. The next exercise—over the page—sets out my attempt at formulating this obvious fact in a way that one might prove.

Then we want to

Exercise 4 *Show that:*

For all valuations v and all formulæ A and all but finitely many t,

$$E_l(A, v, t) = E(A, v) \ \wedge \ E_s(A, v, t) = E(A, v).$$

This is pretty obvious, but it's not totally straightforward to prove. It's certainly obvious that you are going to have to do some induction... but an induction on what? On 't'? Or a structural induction on the subformula relation? This exercise is an object lesson is getting straight quite what it is you are proving by induction, stating the induction hypothesis carefully and being clear in your own mind what kind of induction you are doing.