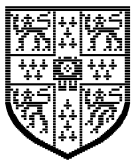

Dr. Thomas Forster
Department of Pure Mathematics and Mathematical Statistics, Centre for Mathematical Sciences, Wilberforce Road, Cambridge CB3 0WB.Tel: +44-1223-337981 (Reception: 765000; fax: 337920.)
email: tf@dpmms.cam.ac.uk. URL: www.dpmms.cam.ac.uk/∼tf. Home: 11 Holyrood Close CB4 3NE, U.K.
mobile in UK +44-7887-701-562; mobile in NZ +64-210580093; mobile in US: +1-412-818-1316.

June 7, 2009

We start with a lot of primitive propositions. Rachel doesn't say whether or not the set of primitive propositions includes the constant symbols '$\bot$' and '$\top$' reserved to evaluate to `false` and `true` respectively. It doesn't sound as if it matters but it might. They are primitive in the sense that we do not (at this stage) analyse their internal structure, and I shall use single letters for these primitive propositions to echo the fact that we are not going to look at their internal structure. They are not primitive in the sense that their truth-values are independent. The propositions are probably attributions of properties to suitably primitive bearers of properties. H (I shall stick to Rachel's notation) is the set of all valuation functions of the set of primitive propositions, a valuation being a function that takes values in {`true`, `false`}. We allow our valuations to be partial, and this is not just for the usual reasons but also because two primitive propositions p and q might be *incommensurable* in the sense that there is no valuation defined on both p and q. If thought of as their graphs, the valuations in H naturally form a poset under $\subseteq$. Rachel doesn't seem to make any assumptions concerning this poset of partial valuations beyond the background assumption that it is not a directed set (which would make everything that follows just the same as the original classical case). Is it chain complete? One would expect so, but nothing seems to turn on it.

The language $\mathcal{L}$ over this alphabet of primitive propositions is built up by means of $\vee$, $\wedge$ and $\rightarrow$, tho' for the moment we are going to hold off defining a negation. There is an obvious recurively defined application function that takes a valuation $h \in H$ and returns a member of {`true`, `false`}. I say 'obvious' but perhaps it might be an idea to spell it out, since the evaluation of complex expressions is subtle when the valuations in play are partial.

The function $\texttt{evaluate}(h, A)$ can be defined by the following Horn clauses

Definition 1.

1. $\texttt{evaluate}(h, A) = \texttt{true} \Rightarrow \texttt{evaluate}(h, A \vee B) = \texttt{true}$;
2. $\texttt{evaluate}(h, B) = \texttt{true} \Rightarrow \texttt{evaluate}(h, A \vee B) = \texttt{true}$;
3. $\texttt{evaluate}(h, A) = \texttt{false} \Rightarrow (\texttt{evaluate}(h, B) = \texttt{false} \Rightarrow \texttt{evaluate}(h, A \vee B) = \texttt{false})$;
4. $\texttt{evaluate}(h, A) = \texttt{false} \Rightarrow \texttt{evaluate}(h, A \wedge B) = \texttt{false}$;
5. $\texttt{evaluate}(h, B) = \texttt{false} \Rightarrow \texttt{evaluate}(h, A \wedge B) = \texttt{false}$;
6. $\texttt{evaluate}(h, A) = \texttt{true} \Rightarrow (\texttt{evaluate}(h, B) = \texttt{true} \Rightarrow \texttt{evaluate}(h, A \wedge B) = \texttt{true})$;
7. $\texttt{evaluate}(h, A) = \texttt{true} \Rightarrow \texttt{evaluate}(h, B \rightarrow A) = \texttt{true}$;
8. $\texttt{evaluate}(h, A) = \texttt{false} \Rightarrow \texttt{evaluate}(h, A \rightarrow B) = \texttt{true}$.

(Here I am using '$\Rightarrow$' for the material conditional in the metalanguage in order not to confuse the reader or myself.)

The fact that these clauses are all Horn means that

Remark 2. *If $h \subseteq h'$ are two valuations and* $\texttt{evaluate}(h, A)$ *is defined, then* $\texttt{evaluate}(h', A)$ *is also defined and is equal to* $\texttt{evaluate}(h, A)$.

Proof: Structural induction on formulæ. ∎

Rachel wants to add to definition 1 a clause that stipulates that if the recursion fails—so that if $\texttt{evaluate}(h, A)$ is not defined—then $\texttt{evaluate}(h, (A \to A)) = \texttt{true}$. By analogy with parallel ideas in computation theory I shall write '$\texttt{evaluate}(h, A)\uparrow$' to mean that the valuation h is not defined at A. This would give us a clause like

9. $\texttt{evaluate}(h, A)\uparrow \ \Rightarrow (\texttt{evaluate}(h, B)\uparrow \ \Rightarrow \texttt{evaluate}(h, A \to B) = \texttt{true})$

If you couch this stuff in the language of three-valued logic it looks much more innocent:

9'. $\texttt{evaluate}(h, A) = \texttt{u} \Rightarrow (\texttt{evaluate}(h, B) = \texttt{u} \Rightarrow \texttt{evaluate}(h, A \to B) = \texttt{true})$

where $\texttt{u}$ is the "middle" truth-value notionally given by h when its output not defined.

In the context of computation theory a move like this is clearly illegitimate, because we do not know when a computation has failed to halt. Here those objections are not good, because conditions like $\texttt{evaluate}(h, A)\uparrow$ are decidable and we can branch on them. Nevertheless one should note that the presence of clauses like *9* above obstruct the proof of the obvious extensions of remark 2. It is for this reason that I prefer formulations like *9* to formulations like *9'*: it makes it much clearer what is really going on.

Rachel has two negations. There is $\neg A$ which has semantics which we can provide in the unproblematic recursive style:

10 $\texttt{evaluate}(h, A) = \texttt{false} \Rightarrow \texttt{evaluate}(h, \neg A) = \texttt{true}$;

11 $\texttt{evaluate}(h, A) = \texttt{true} \Rightarrow \texttt{evaluate}(h, \neg A) = \texttt{false}$.

There is also $\sim A$ which has the above rules plus the non-Horn rule:

12 $\texttt{evaluate}(h, A)\uparrow \ \Rightarrow \texttt{evaluate}(h, \sim A) = \texttt{true}$.

It is not hard to see that $\sim$ obeys excluded middle in the sense that

$$(\forall h \in H)(\forall A \in \mathcal{L})(\texttt{evaluate}(h, (A \vee \sim A)) = \texttt{true})$$

It's slightly harder to see that $\neg$ obeys double negation in the sense that

$$(\forall h \in H)(\forall A \in \mathcal{L})(\texttt{evaluate}(h, (\neg\neg A \to A)) = \texttt{true})$$

0.1 The Lindenbaum Algebra

Next we define a quasiorder (aka preorder) on the set $\mathcal{L}$ of complex propositions by

$$A \leq B \longleftrightarrow_{df} (\forall h \in H)(\texttt{evaluate}(h, A) = \texttt{true} \Rightarrow \texttt{evaluate}(h, B) = \texttt{true}) \wedge (\forall h \in H)(\texttt{evaluate}(h, B) = \texttt{fa}$$

Rachel wanted

$$A \leq B \longleftrightarrow_{df} (\forall h \in H)(\texttt{evaluate}(h, A \to B) = \texttt{true})$$

Quasiorders give rise to equivalence relations, and the equivalence relation corresponding to this quasiorder is

$$A \sim B \longleftrightarrow_{df} (\forall h \in H)(\texttt{evaluate}(h, A) = \texttt{evaluate}(h, B))$$

...and we naturally want to look at the quotient partial order, which will be a kind of generalised Lindenbaum algebra. We establish easily (and Rachel makes a point of this) that meet and join—$\vee$ and $\wedge$—are defined and distribute over each other.