Two brief handouts on semantics for propositional logic

Thomas Forster

November 22, 2012

Contents

1 Compositional Definition of Satisfaction

The key to not getting lost in this enterprise is to bear in mind that the expressions of propositional logic are built up from atomic formulæ (letters) whose meaning is not *reserved* (to use a bit of compsci jargon): they can be anything, but the symbols in the logical vocabulary—'$\wedge$', '$\vee$' and so on—emphatically *are* reserved. We are allowed to assign meanings/truth-vales to the letters and a **valuation** [for propositional language] is a function that assigns truth-values (not meanings!) to the primitive letters of that language. A valuation corresponds to a row in a truth-table. We will use the letter 'v' to range over valuations. Now we define a satisfaction function `sat` that takes valuations and complex expressions and gives back a truth-value. The idea is that `sat` sends a-complex-formula-and-a-valuation to `true` if (and *only* if) the complex formula comes out true in the row of the truth-table corresponding to v.

Definition 1

A complex expression ϕ might be a propositional letter and—if it is—then $\mathtt{sat}(v, \phi)$ *is just $v(\phi)$, the result of applying v to ϕ;*

If ϕ is the conjunction of ψ_1 and ψ_2 then $\mathtt{sat}(v, \phi)$ *is* $\mathtt{sat}(v, \psi_1) \wedge \mathtt{sat}(v, \psi_2)$*;*

If ϕ is the disjunction of ψ_1 and ψ_2 then $\mathtt{sat}(v, \phi)$ *is* $\mathtt{sat}(v, \psi_1) \vee \mathtt{sat}(v, \psi_2)$*;*

If ϕ is the conditional whose antecedent is ψ_1 and whose consequent is ψ_2 then $\mathtt{sat}(v, \phi)$ *is* $\mathtt{sat}(v, \psi_1) \rightarrow \mathtt{sat}(v, \psi_2)$*;*

If ϕ is the negation of ψ_1 then $\mathtt{sat}(v, \phi)$ *is* $\neg\mathtt{sat}(v, \psi_1)$ *;*

If ϕ is the biconditional whose two immediate subformulæ are ψ_1 and ψ_2 then $\mathtt{sat}(v, \phi)$ *is* $\mathtt{sat}(v, \psi_1) \longleftrightarrow \mathtt{sat}(v, \psi_2)$*.*

Notice that here I am using the letters 'ϕ' and 'ψ_1' and 'ψ_2' as variables that range over formulæ, as in the form of words "If ϕ is the conjunction of ψ_1 and ψ_2 then ...". They are not abbreviations of formulæ. There is a temptation to write things like

"If ϕ is $\psi_1 \wedge \psi_2$ then $\mathtt{sat}(v, \phi)$ is $\mathtt{sat}(v, \psi_1) \wedge \mathtt{sat}(v, \psi_2)$"

or perhaps

$$\mathtt{sat}(v, \psi_1 \wedge \psi_2) \;\text{ is }\; \mathtt{sat}(v, \psi_1) \wedge \mathtt{sat}(v, \psi_2) \tag{1}$$

Now although our fault-tolerant pattern-matching enables us to see immediately what is intended, the pattern-matching does, indeed, need to be fault-tolerant. (In fact it corrects the fault so quickly that we tend not to notice the processing that is going on.)

In an expression like '$\mathtt{sat}(v, \phi)$' the 'ϕ' has to be a name of a formula, as we noted above, not a formula or an abbreviation for one. But then how are we to make sense of

$$\mathtt{sat}(v, \psi_1 \wedge \psi_2) \tag{2}$$

The string '$\psi_1 \wedge \psi_2$' has to be the name of formula. Now you don't have to be The Brain of Britain to work out that it has got to be the name of whatever formula it is that we get by putting a '$\wedge$' between the two formulæ named by 'ψ_1' and 'ψ_2'—and this is what your fault-tolerant pattern-matching wetware (supplied by Brain-Of-Britain) will tell you. But we started off by making a fuss about the fact that names have no internal structure, and now we suddenly find ourselves wanting names to have internal structure after all!

In fact there is a way of making sense of this, and that is to use the cunning device of *corner quotes* to create an environment wherein compounds of names of formulæ (composed with connectives) name composites (composed by means of those same connectives) of the formulæ named.. Have a look at `en.wikipedia.org/wiki/Quasi-quotation`

So (1) would be OK if we write it as

$$\mathtt{sat}(v, \ulcorner \psi_1 \wedge \psi_2 \urcorner) \;\text{ is }\; \mathtt{sat}(v, \psi_1) \wedge \mathtt{sat}(v, \psi_2) \tag{3}$$

Corner quotes were first developed in [1]. See pp 33–37. An alternative way of proceding that does not make use of corner quotes is instead to use an entirely new suite of symbols—as it might be '`and`' and '`or`' and so on, and setting up links between them and the connectives '$\wedge$' and so on in the object language so that—for example

$$\psi_1 \;\mathtt{and}\; \psi_2 \tag{A}$$

is the conjunction of ψ_1 and ψ_2. The only drawback to this is the need to conjure up an entire suite of names of connectives, all related suggestively to the connectives they are supposed to name. Thus one might perhaps use

'&' (in lieu of '`and`' as above) as the name for '$\wedge$' There are various difficulties with this. One is that on the whole there are very few obvious candidates: '&' as the name for '$\wedge$' is good, but there is no obvious name for '$\vee$'. Another is the fact that any symbols that are suitably suggestive will also be laden with associations from their other uses, and these associations may not be helpful. Suppose, as I have just suggested, that we were to use an ampersand instead of '`and`'; then the fact that it is elsewhere used instead of '$\wedge$' might cause the reader to assume it is just a synonym for '$\wedge$'. There is no easy way through.

2 Eager and Lazy Evaluation

The recursive (or, as you would say, *compositional*) definition of `sat` in the previous section gives us a way of determining what truth-value a formula receives under a valuation. Start with what the valuation does to the propositional letters (the leaves of the parse tree) and work up the tree. Traditionally the formal logic that grew up in the 20th century took no interest in how things like $\texttt{sat}(v, \phi)$ were actually *calculated*. The recursive definition in the preceding section tells us uniquely what the answer must be but it doesn't tell us uniquely how to calculate it.

The way of calculating $\texttt{sat}(\phi, v)$ that we have just seen (start with what the valuation does to the propositional letters—the leaves of the parse tree—and work up the tree) is called **Eager evaluation** also known as **Strict evaluation**. But there are other ways of calculating that will give the same answer. One of them is the beguilingly named **Lazy evaluation** which we will now describe.

Consider the project of filling out a truth-table for the formula $A \wedge (B \vee (C \wedge D))$. One can observe immediately that any valuation (row of the truth-table) that makes 'A' false will make the whole formula false:

A	$\wedge$	$(B$	$\vee$	$(C$	$\wedge$	$D))$
0		0		0		0
0		0		0		1
0		0		1		0
0		0		1		1
0		1		0		0
0		1		0		1
0		1		1		0
0		1		1		1
1		0		0		0
1		0		0		1
1		0		1		0
1		0		1		1
1		1		0		0
1		1		0		1
1		1		1		0
1		1		1		1

A	$\wedge$	$(B$	$\vee$	$(C$	$\wedge$	$D))$
0	0	0		0		0
0	0	0		0		1
0	0	0		1		0
0	0	0		1		1
0	0	1		0		0
0	0	1		0		1
0	0	1		1		0
0	0	1		1		1
1		0		0		0
1		0		0		1
1		0		1		0
1		0		1		1
1		1		0		0
1		1		0		1
1		1		1		0
1		1		1		1

Now, in the remaining cases we can observe that any valuation that makes 'B' true will make the whole formula true.:

A	$\wedge$	$(B$	$\vee$	$(C$	$\wedge$	$D))$
0	0	0		0		0
0	0	0		0		1
0	0	0		1		0
0	0	0		1		1
0	0	1		0		0
0	0	1		0		1
0	0	1		1		0
0	0	1		1		1
1		0		0		0
1		0		0		1
1		0		1		0
1		0		1		1
1		1	1	0		0
1		1	1	0		1
1		1	1	1		0
1		1	1	1		1

A	$\wedge$	$(B$	$\vee$	$(C$	$\wedge$	$D))$
0	0	0		0		0
0	0	0		0		1
0	0	0		1		0
0	0	0		1		1
0	0	1		0		0
0	0	1		0		1
0	0	1		1		0
0	0	1		1		1
1		0		0		0
1		0		0		1
1		0		1		0
1		0		1		1
1	1	1	1	0		0
1	1	1	1	0		1
1	1	1	1	1		0
1	1	1	1	1		1

In the remaining cases any valuation that makes 'C' false will make the whole formula false.

A	$\wedge$	$(B$	$\vee$	$(C$	$\wedge$	$D))$
0	0	0		0		0
0	0	0		0		1
0	0	0		1		0
0	0	0		1		1
0	0	1		0		0
0	0	1		0		1
0	0	1		1		0
0	0	1		1		1
1	0	0	0	0	0	0
1	0	0	0	0	0	1
1		0		1		0
1		0		1		1
1	1	1	1	0		0
1	1	1	1	0		1
1	1	1	1	1		0
1	1	1	1	1		1

A	$\wedge$	$(B$	$\vee$	$(C$	$\wedge$	$D))$
0	0	0		0		0
0	0	0		0		1
0	0	0		1		0
0	0	0		1		1
0	0	1		0		0
0	0	1		0		1
0	0	1		1		0
0	0	1		1		1
1	0	0	0	0	0	0
1	0	0	0	0	0	1
1	0	0	0	1	0	0^*
1	1	0	1	1	1	1^*
1	1	1	1	0		0
1	1	1	1	0		1
1	1	1	1	1		0
1	1	1	1	1		1

The starred '0^*' and '1^*' are the only cases where we actually have to look at the truth-value of D.

References

[1] W.V. Quine, Mathematrical Logic