CS 5860 Sept 15, 2011

Lecture 6 Formal Logics continued, Computational Logic

Please read about Tableau Proofs from the pages of Smullyan posted with Lecture 5. You can see these proofs as "systematic searches" for counter examples to a proposed theorem. These proofs are closely related in form to those we consider in this lecture – in the supplemental printed material.

Notice that we see many different notations for Boolean expressions and propositional formulas. In Nuprl there is one underlying uniform term structure and then many display forms for making the expressions more readable. The uniform notation has this format

operator name op

possible subterms $op(t_1; t_2; \cdots; t_n)$

In this format the Boolean expressions are: $not(t_1)$ $and(t_1; t_2)$ $or(t_1; t_2)$

We display these in various ways such as

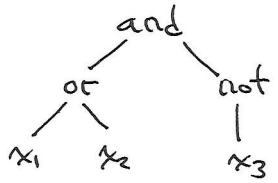
$not(t_1)$ as: $\sim t_1$ $(not\ t_1)$ (the tilde, $\sim$, is due to Russell)

$or(t_1; t_2)$ as: $(t_1 \vee t_2)$ $(t_1 \vee t_2)$ (the $\vee$ comes from Latin vel, "or")

$and(t_1; t_2)$ as: $(t_1\ and\ t_2)$ $(t_1 \& t_2)$ (& is a ligature for the Latin et for and)

We display $or(not(t_1); t_2)$ also as $t_1 \supset t_2$ ($\supset$ is due to Peano who also wrote $t_2 \subset t_2$, read as t_2 a consequence of t_1)

We can also display expressions as trees $and(or(x_1; x_2); not(x_3))$ as



CS 5860
Lecture 6

For Boolean logic the basic type is $\mathbb{B} = \{t, f\}$ the Booleans.
For Computational logic the basic type is $\mathbb{P}$ (also Prop)
for propositions. Propositions are a more subtle and substantive idea than the notion of a truth value.

Intuitively a proposition is an expression that we can assert or judge to be established or known. A proposition P is asserted by an agent who claims to know it. Here are examples.

2 is an even prime number.
there are an unbounded number of primes.
Petersen's Mutual Exclusion protocol is safe.
$7 = 2*3+1$ euclid(7,3) = (2,1)

Here is a sentense that seems to be a proposition but is not: This sentence is false.

Before we can assert an expression we must know that it is a proposition. We indicate this by writing

$P \in Prop$ or P in Prop

We can declare a variable to be a propositional variable by writing x : Prop.

CS 5860

Lecture 6 Computational Logic

We can know that P is a proposition and not have any idea whether or not it is true, but because it is a proposition we will understand how we can know it. Here are some famous propositions which have not been established. We say they are unsettled or open or unknown. or undecided.

Bexp-Sat $\in$ PTime SAT $\in$ PTime

(we also say Bexp-Sat $\in$ P, etc.)

Goldbach's conjecture (1742) Let $\mathbb{E}$ be the even numbers.

$$\forall x : \mathbb{E}\ (x \geq 6 \Rightarrow \exists p_1, p_2 : \mathbb{N}.\ (x = p_1 + p_2 \ \&\ \text{OddPrime}(p_1) \ \&\ \text{OddPrime}(p_2)))$$

Prime Pairs Conjecture

$$\forall n : \mathbb{N}.\ \exists p : \mathbb{N}.\ (n < p \ \&\ \text{Prime}(p) \ \&\ \text{Prime}(p+2))$$

Very very big sets exist

ZFC + n-huge cardinals is consistent

The set theory on which the Journal of Formalized Mathematics is based, Tarski-Grothendeick Set Theory (TG Set Theory) is consistent.

Note, we know these examples of $A \Rightarrow B$ for propositions A and B without knowing either A or B, e.g. A and B are open, unknown.

SAT $\in$ P $\Rightarrow$ Bexp-Sat $\in$ P

Extended Riemann Hyp $\Rightarrow$ SAT $\in$ P

(We will not discuss the Extended Riemann Hyp (ERH), but you can read about it on Wikipedia.)

CS 5860
Lecture 6

What does it mean to know $A \Rightarrow B$ when A and B are open (undecided)? We must be able to know this without knowing the truth values. It must follow from knowing the meaning of A and B, but at the level of propositional logic, A and B can be abstract, say just propositional variables.

Can we abstract from the specific examples to say what $(A \Rightarrow B)$ means for arbitrary propositions?

The way computer scientists and constructive mathematicians do this is to say that $(A \Rightarrow B)$ is known when there is a computational process, say a computable function f which converts evidence for A (or the meaning or concept or cause of A) into evidence for B, the reason for B, its "truth-maker."

To make sense of this at the level of abstract propositions, we must say what the evidence is for propositions built from atomic propositions using and, or, implies.

We deal with negation, $\neg A$, separately. It is a much harder idea than the Boolean negation.

Turn now to the notes on Evidence Semantics included on the Web page for Lecture 6.