Correctness of $SC$ (43)

**Agreement**

$\forall i, j : < n, n' > : n^+ \forall b, b' : B. \\
(\exists e_i, e_j : i \neq j \Rightarrow \text{Decide}(e) = < n, b > \land \text{Decide}(e') = < n', b' >) \\
\Rightarrow (n = n' \Rightarrow b = b')$

We did this proof informally before and it is in the lecture notes for Tue Nov 29.

**Validity**

$\forall n : n^+ \forall b : B. (\exists e_i : \text{Decide}(e) = < n, b > \Rightarrow \\
\exists e_i : e < e \land \text{Propose}(e') = < n, b >)$

We can show by induction that the only values voted on are values proposed. It is a good exercise to carry out this proof in the logic of Events.

**Extra Credit:** Turn in a proof of this theorem with your final project due Dec 7 at the end of the day.
In Event ML we can describe protocols using Mark Bickford's notion of event combinators. This is a very powerful formalism described in the article Introduction to Event ML which is available on the course web page where it will be updated until it is posted at www.nuprl.org, in due course. Here is roughly what the SC protocol looks like in this notation. The actual combinators and protocol are given in the above mentioned article in Figure 5, section 3.5.

\[
\text{SC-Replica e locs}
\]

\[
\begin{align*}
\text{Replica} & \equiv \text{New Proposal} \gg \text{Voter} \\
\text{Voter}(n,c) & \equiv \text{Round}(c) \uparrow \text{Notify} \uparrow \text{New Round} \gg \text{Round}(a,c) \\
\text{Round}(b,c) & \equiv \text{SendVotes}(n,c) \uparrow \text{Quorum}
\end{align*}
\]

\text{Constructive Fischer- Lynch - Paterson}

See my article on the course web page. This result shows that by proving that a consensus protocol is non-blocking, we create a possible undefeated attacker via denial of service.

\text{Definition: a deterministic consensus procedure P is called effectively non-blocking if from any reachable global state s of the execution of P and any subset Q of n-t of the n processes executing P which have not failed we can find an execution } \alpha \text{ from s using Q and a process } p_x \text{ in Q which decides a value. We discuss this for the case of Boolean values.}
Theorem: Given any deterministic effectively nonblocking consensus procedure \( P \) with more than two processes and tolerating a single failure, we can effectively construct a nonterminating execution of it (by controlling the environment).

As a direct corollary of this theorem, we can prove the well-known FLP theorem because if we are given a purported terminating consensus procedure, it is a witness to its own effective nonblocking!

Corollary: There is no single-failure responsive consensus algorithm.

Note: Responsive is another term for valid, and to say it is an algorithm is to say that it halts on all admissible runs.
Another interesting limiting result about distributed computing is by Joe Halpern (Cornell CS) and Yoram Moses. They show that in a distributed system where there is no bound on communication time, it is impossible for a group of processes to achieve common knowledge. This is an instance of the coordinated attack problem discussed by Jim Gray in 1978. (Jim was a Turing Award winner who mysteriously disappeared while sailing alone in the Pacific Ocean off San Francisco.) The problem is described in the supplemental readings. I'll describe it briefly in the lecture.

Closing Themes

Proofs-as-Processes: We have seen how it is possible to derive distributed protocols from very high level descriptions using the event combinators of Event Logic. This extends our theme of proofs-as-programs to a new and important class of problems, and extends the powerful formal method of correct-by-construction programmes.

Computational logics and computational evidence semantics are the foundations of correct-by-construction programmes. Dependent types are key to evidence semantics and will become features of some programming languages and logical programming environments.