CS5860

Thur. Dec 1, 2011 Correctness of Simple Consensus (SC)

PLAN

- Correctness of simple consensus
  - agreement
  - validity

  event combinators in Event NKL
- Constructive Fischer-Lynch-Paterson (FLP)

  Impossibility of common knowledge (Halpern-Moses)
  can't attain common knowledge
  what knowledge are processes "born with"
  type theory as common knowledge
- Closing themes

Correctness of SC (2/3)

Agreement:

$$\forall i,j : G.\ \forall n,n' : \mathbb{N}^+\ \forall b,b' : \mathbb{B}.$$
$$\left(\exists e \in i,\ e' \in j.\ i \neq j \ \&\ Decide(e) = \langle n,b\rangle \ \&\ Decide(e') = \langle n',b'\rangle\right)$$
$$\Rightarrow (n = n' \Rightarrow b = b')$$

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

Validity $\forall n : \mathbb{N}^+\ \forall b : \mathbb{B}.\ (\exists e \in i.\ Decide(e) = \langle n,b\rangle \Rightarrow$
$\exists e' \in i.\ e' < e\ \&\ Propose(e') = \langle n,b\rangle)$

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.

CS 5860

Dec. 1, 2011 Event Combinators

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.

SC-Replica @ locs

Replica == NewProposal >>= Voter

Voter(n,c) == Round(0) || Notify || NewRound >>= Round(n,c)

Round(n,c) == SendVotes(n,c) || Quorum

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 undefeatable attacker, via denial of service.

Definition: a deterministic consensus procedure P is called effectively nonblocking 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 α from s using Q and a process P_x in Q which decides a value. We discuss this for the case of Boolean values.

CS 5860

Thur. Dec 1, 2011 Constructive FLP Theorem

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.

CS 5860

Thur Dec 1 Common Knowledge and Closing Themes

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 reading. 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 programming.

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