CS3110 Spring 2016 Lecture 25: Event Logic

Robert Constable

Topics

1. Reasoning about 2/3 consensus.

   We will prove three critical properties of consensus protocols. We discussed these properties in Lecture 22. They are:

   P1: Unanimous inputs of value v cause the protocol to decide on v. We call this *unanimity*.

   P2: All decided values are inputs. We call this *validity*.

   P3: Any two decisions have the same value. We call this *agreement*.

2. The Logic of Events.

   We can axiomatize and formally prove the above properties of protocols as well as others. The reasoning is based on events in the (general process) computing model. We can axiomatize this logic with a few intuitively clear axioms that capture features of the *message sequence diagrams*.

3. We will review the *enduring concepts* from the course next lecture. The great programming languages leave their mark on computing theory as well as on decades of important applications. The earliest major languages were Fortran, Algol, and Lisp – from the 1960s approximately. Fortran and Lisp are still well used, and Algol left its make on the theory and methodology of creating and implementing programming languages.

1 Reasoning about 2/3 (simple) consensus

The Synthesized 2/3 Protocol

Begin
r : Nat
decided_i, vote_i : Bool

$r = 0$, L = nil
decided_i = false
vi = input to Pi
vote_i = vi

Until decided_i **do**:

1. $r := r + 1$
2. **Broadcast** vote <r,vote_i> to group G
3. **Collect** $2f + 1$ round r votes in list L
4. vote_i := majority(L)
5. **If** unanimous(L) **then** decided_i:= true

End
Broadcast decision

Note: Collect means to add any received votes to L, and check whether there are $2f + 1$ votes with the same round number.

We will prove the three properties of the protocol that we discussed previously: *unanimity, validity*, and *agreement*.

P1: If all inputs propose value v, then any decision must have value v. We wrote this symbolically as:

$\forall v : T.(\forall e : \text{Event}.\ \mathit{input}(e) = v) \Rightarrow (\forall e : \text{Event}.\ \mathit{decide}(e) = v)$

We prove this by induction on rounds. In round 1 only inputs are broadcast to the replicas. Hence only inputs are collected at step 3. Thus only inputs are counted in the majority. So only inputs are broadcast in round 2. This argument also shows that if only inputs are broadcast in

round r, then only inputs can be decided. If the only input is v, then this is the only possible decision. QED

P2: All decided values are input values.

$$\forall e : \text{Event}.\ E(decide(e)) \Rightarrow \exists e' : \text{Event}.\ (e' < e\ \&\ decide(e) = input(e'))$$

We can also prove this by induction on rounds. If the protocol is in round 1, then all collected values are inputs. For any round r, if all collected values are inputs, then only inputs will be broadcast. Hence any unanimous value will be an input. QED

P3. Agreement, any two decisions have the same value.

$$\forall e_1, e_2 : \text{Event}.\ E(decide(e_1))\ \&\ E(decide(e_2)) \Rightarrow decide(e_1) = decide(e_2)$$

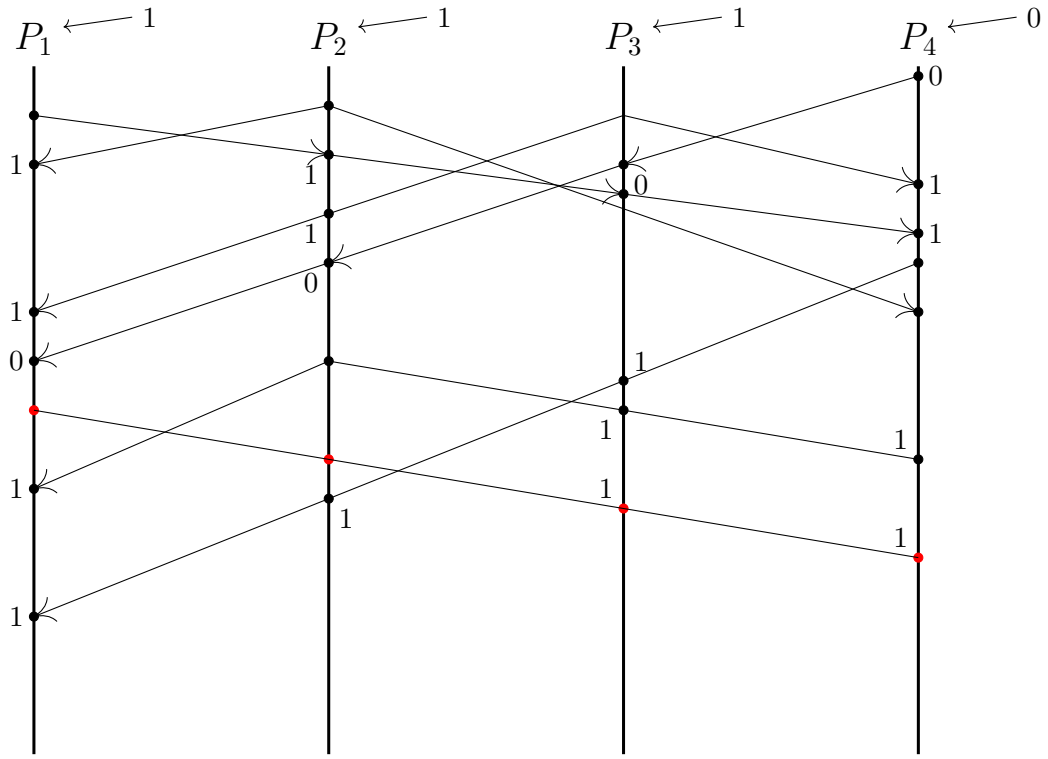
Suppose that two processes execute *decide* in the same round r, e.g. set $decided\_i :=$ true and $decided\_j :=$ true. This means that they have collected $2f + 1$ votes and they are unanimous at process i say for a value v_i. (For example, for $f = 1$, 3 votes are collected, and all three are for v_i.) In process j, 3 values are collected, suppose they are all for $v_j \neq v_i$. This is impossible since there are only 4 votes to collect and three are for v_i.

In general, we see that the requirement that a decision is made only when $2f + 1$ votes are the same, disallows two different collections of $2f + 1$ votes since there are only $3f + 1$ possibilities, not $4f + 2$. QED

If they decide in different rounds, say i in r and j in $r' > r$, then we need a more subtle argument. **Optional exercise**: Finish the proof for this case.

2 The Logic of Events

One way to understand asynchronous distributed protocols is to think in terms of *message sequence diagrams* as we did in explaining the 2/3 consensus protocol. Here is a diagram to remind us for the case of one possible failure, $3f + 1$ will have the value 4. So we show the four participants and the inputs they receive.



The *events* are marked as dots on the "time line" of each process. The logic of events postulates properties of the events. At each location, we see that the events are linearly ordered.

There is another important ordering relation less clearly discernible. Lamport called it out and named it as *causal order*. The red event at P_1, the 5th event on the P_1 time line, was *caused by* the four previous events at P_1. Using the message sequence diagram, we can trace out the causal order relation.

The logic of events provides axioms to characterize the local order of events at each process as well as the causal order relation. The logic also depends on assumptions about message delivery and process failures.

The logic of events has been fully implemented by the Nuprl proof assistant and used to prove that protocols have certain behaviors under various assumptions about process failure and message delivery. The current list of axioms and theorems can be examined in the Nuprl mathematics library at `http://www.nuprl.org/wip/EventOrdering/new!event-ordering/index.html` . We discuss two key properties of this logic.

Property 1:	The events at any location are totally ordered.
Property 2:	If only the events at finitely many locations are considered, then causal order is well founded. *This means that we can prove properties of protocols by induction on causal order.*

Here are some of the other "laws" of the logic of events.

Table 1. Signatures in the Logic of Events

Events with Order	**Definitional extensions**
E: $\mathbb{D}$	**loc:** $E \to Loc$
pred?: $E \to (E + Loc)$	**first:** $E \to Bool$
sender?: $E \to (E + Unit)$	**isrcv:** $E \to Bool$
	$x < y\,, \quad x <_{loc} y$
and with Values	
Kind $= Loc \times Act + Lnk \times Tag$	**sender:** $\{e : E \vert isrcv(e)\} \to E$
vtyp: $Kind \to Type$	**link:** $\{e : E \vert isrcv(e)\} \to Link$
kind: $e : E \to Kind$	**tag:** $\{e : E \vert isrcv(e)\} \to Tag$
val: $e : E \to vtyp(kind(e))$	
and with State	
typ: $Id \to Loc \to Type$	**state**$(i) = x : Id \to typ(x, i)$
initially: $x : Id \to i : Loc \to typ(x, i)$	
when: $x : Id \to e : E \to typ(x, loc(e))$	**state-when:** $e : E \to state(loc(e))$
after: $x : Id \to e : E \to typ(x, loc(e))$	**state-after:** $e : E \to state(loc(e))$

Table 2. Axioms of Basic Event Structures

1. On any link, an event sends boundedly many messages; formally: $\forall l : Link.\forall e : E.\exists e' : E.\forall e'' : E.R(e'', e) \Rightarrow e'' < e' \wedge loc(e') = dst(l)$ where $R(e'', e) \equiv isrcv(e'') \wedge sender(e'') = e \wedge link(e'') = l$
2. The predecessor function is one-to-one; formally: $\forall e_1, e_2 : E.\ pred?(e_1) = pred?(e_2) \Rightarrow e_1 = e_2$
3. Causal order is (strongly) well-founded; formally: $\exists f : E \rightarrow \mathbb{N}.\forall e_1, e_2 : E.\ e_1 < e_2 \Rightarrow f(e_1) < f(e_2)$
4. The location of the sender of an event is the source of the link on which the message was received; formally: $\forall e : E.\ isrcv(e) \Rightarrow loc(sender(e)) = src(link(e)$
5. Links deliver messages in FIFO (first in first out) order; formally: $\forall e_1, e_2 : E.\ link(e_1) = link(e_2) \Rightarrow sender(e_1) < sender(e_2) \Rightarrow e_1 < e_2$
6. State variables change only at events, so that: $\forall e : E.\neg\ first(e) \Rightarrow (x$ **when** $e) = (x$ **after** $pred(e))$

Message sequence diagram for 2/3 consensus