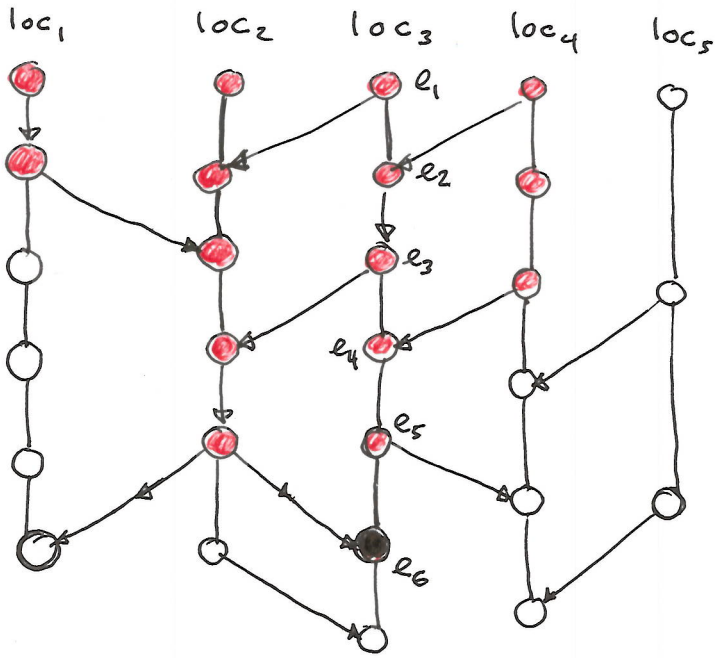
CS 5860

Thur Nov 17, 2011 A Theory of Events in First-Order Logic

Last time we defined the very important causal order relation, $e_1 < e_2$. It is the transitive closure of Pred(x,y). Now we will show that this is a decidable relation, i.e.

$$\forall x{:}E.\forall y{:}E.((x<y) \vee \sim(x<y))$$

We first look closely at the possible structure of this ordering and review its definition. We use message sequence diagrams




The red events show those which are causally before $e_6$ at $loc_3$. The arrows (⟶) indicate messages are sent, so the target of the arrow is a receive event. The lines without arrows, (—), say from $e_5$ to $e_6$ at $loc_3$ shows a simple predecessor relation, event $e_5$ might change the state at $loc_3$ and can affect what happens after $e_6$.

CS5860

Thur Nov 17, 2011 A Theory of Events in FOL

We can define the predecessor events at a location as a list, say at $loc_3$ the predecessor events of $e_6$ are, so the list $before(e_6) = [e_5, e_4, e_3, e_2, e_1]$. The events causally before an event form a tree which we call the cone of the event.

Definition

$$before(e) = \underline{if}\ first(e)\ \underline{then}\ [e, nil]\ \underline{else}\ cons(e, before(pred(e)))$$

where $cons(e, [e_1, \cdots, e_n, nil]) = [e, e_1, \cdots, e_n, nil]$.

Definition

$$cone(e) =$$

$$\underline{if}\ rcv?(e)\ \underline{then}\ \underline{if}\ first(e)\ \underline{then}\ \langle e, loc(e), cone(sender(e))\rangle\ \underline{else}\ \langle e, cone(pred(e)), cone(sender(e))\rangle$$

$$\underline{else}\ \underline{if}\ first(e)\ \underline{then}\ \langle e, loc(e), nil\rangle\ \underline{else}\ \langle e, cone(pred(e)), nil\rangle$$

One way to decide whether $e_1 < e_2$ is to form $cone(e_2)$ and then check to see if $e_1 \in cone(e_2)$ using the fact that equality of events is decidable.

Another way is to prove it by induction of causal order.

Theorem $\forall x, y : E.\ (x < y) \vee \sim(x < y)$ – causal order is decidable.

CS 5860

Thur Nov 17, 2011 A Theory of Events in FOL continued

The induction principle for causal order is this

Axiom $\forall x{:}E.(\forall y{:}E.\ (y<x \Rightarrow P(y)) \Rightarrow P(x)) \Rightarrow \forall x{:}E.\ P(x)$

This is like complete induction on $\mathbb{N}$, recall

$$\forall n((\forall y.(y<n \Rightarrow P(y)) \Rightarrow P(n)) \Rightarrow \forall x.\ P(x)$$

See the supplemental notes for Tue Nov 15 for an argument that causal order is strongly well founded, given in the techical report A Causal Logic of Events in Formalized Computational Type Theory.

In that article we do not discuss the idea of a choice function $f$ that maps $E$ to natural numbers $\mathbb{N}$. Thus that account does not take into consideration the kinds of nondeterminism that arise in real word systems. Such an account depends on some mechanism to introduce choice or uncertainty in the model. This choice arises from not knowing in advance the arrival order of messages destined for a location. We can conduct a "tour" of event orderings only after the fact. We do not have time in this course to discuss the mechanisms for non determinism in detail. We will however examine the notion in more detail when we study consensus protocols next week. We will sketch the idea of choice sequences.

CS 5860

Thur. Nov 17, 2011 A Theory of Events in FOL continued.

Next we examine the notion of state at a location. We imagine that processes at $loc_i$ have access to local state accessible by identifiers, another basic sort of the theory denoted $Id(x)$. The state stores data or values. We say

$$\forall i, x, (Loc(i) \,\&\, Id(x) \Rightarrow \exists v.\ Value(v) \,\&\, St(i, x, v))$$

We assume only finitely many identifiers for the examples we consider, denoted $x, x_1, x_2, \ldots$ We can specify the initial values $x$ initially $i$ is the initial value of identifier $x$ at location $i$.

We also introduce the temporal operators

$x$ when $e$
$x$ after $e$

Axiom $\forall e: E.\ \neg first(e) \Rightarrow (x \text{ when } e) = (x \text{ after } pred(e))$

Define $x \,\Delta\, e$ iff $x$ after $e \neq x$ when $e$
This is the change operator.

CS 5860

Thur Nov 17, 2011 A Theory of Events in EOL continued

## Message Automata Realizers for event statements

1. @i p(x initially i) realizes $\forall e \in @i.\ first(e) \Rightarrow p(x \text{ when } e)$

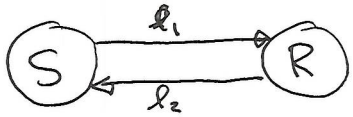
2. @i $rcv_\ell(v)$ effect $x := f(s,v)$ for s the state at i
realizes
$\forall v: Value.\ \forall \ell: Link.\ \forall e \in @i.\ e = rcv_\ell(v) \Rightarrow (x \text{ after } e) = f(s,v)$

3. @i $send_\ell(v)$ realizes
$\forall e \in @i.\ (kind(e) = send_\ell(v)) \Rightarrow \exists e' \in @dest(\ell).\ kind(e') = rcv_\ell(v) \ \&\ sender(e') = e$

4. @i only L affects x for L a list of actions. realizes
$\forall e \in @i.\ kind(e) \notin L \Rightarrow \neg(x \Delta e) \ \&\ (x \Delta e \Rightarrow kind(e) \in L)$
This is called a frame condition.

5. @i only L sends $\langle v, tag \rangle$ realizes
$\forall e \in @dest(\ell).\ kind(e) = rcv_\ell(\langle v, tag \rangle) \Rightarrow kind(sender(e)) \in L.$
This is called a sends frame condition.

Now we are prepared to treat the acknowledgement protocol from Nov 10 in more detail. Recall the context.




S and R are processes linked by FIFO communication channels $\ell_1, \ell_2$.

As an example, suppose we want a system of processes $\mathcal{P}$ with the property that two of its processes, say $S$ and $R$ connected by link $\ell_1$ from $S$ to$R$ and $\ell_2$ from $R$ to $S$ should operate using explicit acknowledgement. So when $S$ sends to $R$ on $\ell_1$ with tag $tg$, it will not send again on $\ell_1$ with this tag until receiving an acknowledgement $tag$, $ack$, on $\ell_2$. The specification can be stated as a theorem about event structures arising from extensions $mathcal(P)'$ of $\mathcal{P}$, namely:

**Theorem 1** *For any distributed system $\mathcal{P}$ with two designated processes $S$ and $R$ linked by $S \xrightarrow{\ell_1} R$ and $R \xrightarrow{\ell_2} S$ with two new tags, tg and ack, we can construct an extension $\mathcal{P}'$ of $\mathcal{P}$ such that the following* ***specification*** *holds:* $\forall e_1, e_2 : E.loc(e_1) = loc(e_2) = S \;\&\; kind\ (e_1) = kind(e_2) = send(\ell_1, tg).\ e_1 < e_2 \Rightarrow \exists r : E.\ loc(r) = S \;\&\; kind\ (r) = rcv(\ell_2, ack).\ e_1 < r < e_2$.

This theorem is true because we know how to add clauses to processes $S$ and $R$ to achieve the specification, which means that the specification is constructively *achievable*. We can prove the theorem constructively and in the process define the extension $\mathcal{P}'$ implicitly. Here is how.

**Proof**: What would be required of $\mathcal{P}'$ to meet the specification? Suppose in $\mathcal{P}'$ we have $e_1 < e_2$ as described in the theorem. We need to know more than the obvious fact that two send events occurred namely $< tg, m_1 >, < tg, m_2 >$ were sent to $R$. One way to have more information is to remember the first event in the state. Suppose we use a new Boolean state variable of $S$, called $rdy$, and we require that a send on $\ell_1$ with tag $tg$ happens only if $rdy = true$ and that after the send, $rdy = false$. Suppose we also stipulate in a frame condition that *only a receive on $\ell_2$* sets ready to true, then we know that $rdy$ **when** $e_1 = true$, $rdy$ **after** $e_1 = false$ and $rdy$ **when** $e_2 = true$. So between $e_1$ and $e_2$, some event $e'$ must happen at $S$ that sets $rdy$ to true. But since only a $rcv(\ell_2, ack)$ can do so, then $e'$ must be the receive required by the specification.

This argument proves constructively that $\mathcal{P}'$ exists, and it is clear that the proof shows how to extend process $S$ namely add these clauses:

$a$ : **if** $rdy = true$**then**
    $send(\ell_1, < tg, m >); rdy := false$
$r$ : **rcv**$(\ell_2, ack)$ **effect** $rdy := true$
    $only[a, r]$ **affect** $rdy$

**QED**

We could add a liveness condition that a send will occur by initializing $rdy$ to true. If we want a live dialogue we would need to extend $R$ by.

**rcv**$(\ell_1, < tg, m >)$ **effect send** $(\ell_2, ack)$

but our theorem did not require liveness.