Remember Final Exam Monday May 14, 2012 Upson 315 9:30am - 12:30am

Note, be prepared to chose whether you will answer a simple question on partial types or a simple question on event logic on the final. For partial types the question will be about modeling IMP or your semantics of recursive procedures. For event logic it will be about properties of the event structures generated by Message Automata. The questions will be based on Lect 39 and 40 and the supplemental notes on Event Structures with Lecture 39.

Event Structures

Event structures arise from the execution of a distributed system over time. The system starts at a finite number of locations. In some models, an action can create new locations initialized with processes, e.g. in the Nuprl General Process Model (GPM) - see www.nuprl.org Publications 2010 "Generating Event Logics ...." The event structures arise from the interaction of processes with the environment or the communication system. The events are the actions of receiving and sending messages and the local actions of the processes. The messages convey typed information.

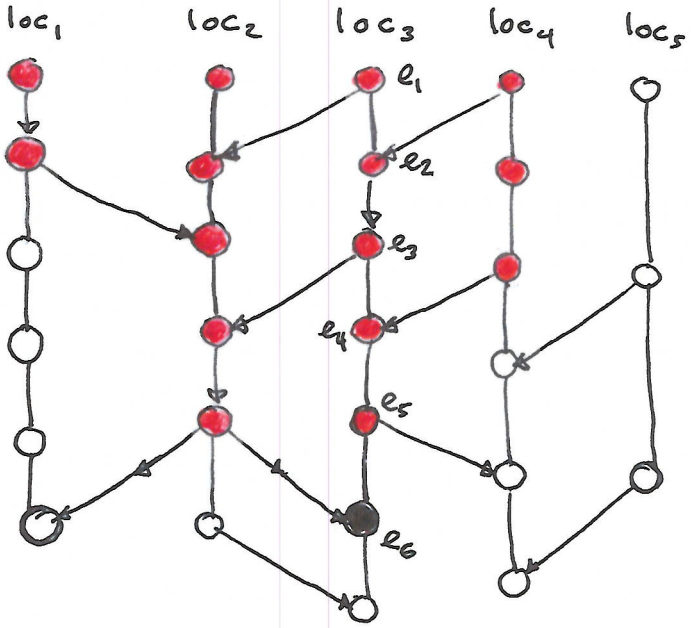
We capture the general properties common to all event structures using the formulas of Event Logic. We will examine these common properties informally and refer to the Lecture 39 supplemental notes.

Event structures continued

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.

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

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.

The "event cone"

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

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

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

Definition

$$cone(e) =$$

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

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

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 on causal order.

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

Next we need an axiom for finding the sender of an event. If the event is not a receive, then we associate the unit value rather than the sender.

Axiom

$\forall x : E.\exists y.(E(y)$ & $\texttt{Sender}(x,y)) \vee (\texttt{Unit}(x)$ & $\texttt{NotReceive}(x))$

The realizer is the term $\texttt{sender?}(x)$. The term computes by finding the canonical form of the event. If it has the form $\texttt{rcv}(v)$ then we find the sender from the header or the channel (as with pred?). If x is not a receive, then $\texttt{sender?}(x)$ computes to the unit value $\bullet$.

Now we will define $x \lhd y$, also written as the binary relation $\texttt{Pred}(x,y)$. First we define these functions and predicates.

```
first?:E → Bool
first?(x)  = spread(pred?(x);y,p. decide(p;l. false; r. true))
           = let pred?(x) = (y,p) in if isl(p) then false
                                                else true
sender?: E → E + Unit
  rcv?(x)  = decide(sender?(x); l. true; r. false)
           = if isl(sender?(x)) then true else false.
 First(x) iff first?(x) = true
   Rcv(x) iff rcv?(x) = true
Pred(x,y) iff (¬ First(y) & x = pred(y))
              ∨ (Rcv(y) & x = sender(y))
thus on y such that ¬ First(y),
pred(y) = spread(pred?(y);x,p.x)
and on y such that rcv(y),
sender(y) = if isl(sender?(y)) then outl(y)                    .
```

We will now form the *transitive closure* of $\texttt{Pred}(x,y)$, this will be Lamport's *causal order* relation, $x < y$. We will be able to prove $\forall x,y : E.((x < y) \vee \sim (x < y))$, but we need more axioms.

Given a relation $\texttt{R}(x,y)$ we define its *transitive closure* as follows. Define $\texttt{R}^{(0)}(x,y)$ $\texttt{iff}$ $\texttt{R}(x,y)$ and $\texttt{R}^{(n+1)}(x,y)$ $\texttt{iff}$ $\texttt{R}(x,z)$ & $\texttt{R}^{(n)}(z,y)$ for some z.

$$R^*(x,y) \texttt{ iff } \exists n : \mathbb{N}.R^{(n)}(x,y).$$

Using the notation $x \lhd y$ for $\texttt{Pred}(x,y)$, here is the same definition. Define $x \lhd^{(0)} y$ $\texttt{iff}$ $x \lhd y$ and $x \lhd^{(n+1)} y$ $\texttt{iff}$ $\exists z.(x \lhd z$ & $z \lhd^{(n)} y.)$.

Say $x \lhd^* y$ $\texttt{iff}$ $\exists n : \mathbb{N}.x \lhd^{(n)} y$.

Definition: Lamport's *causal order on events* is $\texttt{Pred}^*$ (same as $\lhd^*$).

We will show that we can reason by induction on $\texttt{Pred}^*$. An elegant way to do this is by postulating that $\texttt{Pred}(x,y)$ is strongly well founded.

Axiom

$\texttt{Pred}(x,y)$ is *strongly well founded*, i.e. there is a "choice sequence f" from E to $\mathbb{N}$ such that $\forall e,e' : E.\ \texttt{Pred}(e,e') \Rightarrow f(e) < f(e')$.

We also need an axiom about $\texttt{pred}(x)$.

Axiom

The predecessor function, pred, is injective (i.e. one-to-one).
$\forall e,e' : E.(\texttt{loc}(e) = \texttt{loc}(e')$ & $\neg\texttt{First}(e)$ & $\neg\texttt{First}(e')) \Rightarrow$
$(\texttt{pred}(e) = \texttt{pred}(e')) \Rightarrow e = e'$.

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

$$\forall i, x_0 \, (Loc(i) \,\&\, Id(x) \Rightarrow \exists v. \; Value(v) \,\&\, St(i, x, v))$$

We assume only finitely many identifiers for the examples we consider, denoted $x, x_1, x_2, \cdots$. 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.

Message Automata Realizers for event statements

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

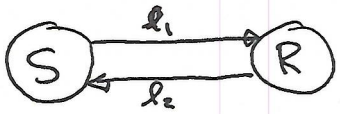
2. @i $rcv_\ell(v)$ effect $x := f(s,v)$ for s the state at i
   realizes
   $\forall v: Value.\ \forall \ell: Link.\ \forall e \in i.\ e = rcv_\ell(v) \Rightarrow (x \text{ after } e) = f(s,v)$

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

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 Nor 10 in more detail. Recall the context.



S and R are processes linked by FIFO communication channels ℓ_1, ℓ_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 ℓ_1 from S to$R$ and ℓ_2 from R to S should operate using explicit acknowledgement. So when S sends to R on ℓ_1 with tag tg, it will not send again on ℓ_1 with this tag until receiving an acknowledgement tag, ack, on ℓ_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 ℓ_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 ℓ_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:

$$
\begin{array}{l}
a : \textbf{if } rdy = true \textbf{then} \\
\quad send(\ell_1, < tg, m >); rdy := false \\
r : \textbf{rcv}(\ell_2, ack) \textbf{ effect } rdy := true \\
\quad only[a, r] \textbf{ affect } rdy
\end{array}
$$

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.

$$\textbf{rcv}(\ell_1, < tg, m >) \textbf{ effect send } (\ell_2, ack)$$

but our theorem did not require liveness.