Event Systems
by Robert L. Constable

PRL Seminar - September 29, 2003
Cornell University, Dept. of Computer Science

This lecture is based on work done with Mark Bickford.

Originally Lecture 4 of a series of lectures given at the Marktoberdorf Summer School 2003
Formal Methods

One of the major research challenges faced by computer science is providing a technology for building highly reliable software systems that perform well and evolve economically to ever-higher reliability and better performance.

One approach is using formal methods, tools based on logic. They are not widely adopted. There are several approaches to improving them:

- More fully automated tools – type checkers, model checkers, decision procedures
- More effective interactive tools – provers
Our Experience

We have worked on distributed system verification for a decade, with Ken Birman’s group at Cornell.

We discovered that we have accumulated sufficient knowledge that we could participate in protocol design and development at the speed of the designers and programmers.
Concurrency Models

There is an enormous literature providing models for concurrent computation and distributed systems, e.g. Petri nets, process calculi, operational semantics, domain theories, machine models (IO Automata, message automata), model theory, etc.

These models support several logics of processes, e.g. temporal logics, programming logics, etc.

Our work has been based on variants of Nancy Lynch’s IO Automata. We presented some results in this summer school in 1999, with Jason Hickey.
Stack Model

Sender

Layer 1
Layer 2
Layer 3
Layer 4
Bottom Layer

Protocol Stack

Event

Receiver

Layer 1
Layer 2
Layer 3
Layer 4
Bottom Layer

Protocol Stack

Message

Header

NET

FIFO Queues

Layer
Event Systems - Key Concepts

- Type E of events - abstract
- Type Loc of loci of events or process identifiers
- Type Lnk of communication links
- There are two kinds of events: (1) local actions and (2) receives; each can also send on links
- Each receive event has a unique sender
- Events at any locus are totally ordered, and causal order relates events among loci
Example - Two-Phase Handshake

\[ E_p = \{ e : E | \text{loc}(e) = p \} \]

\[ \text{Snd}_{p,l} = \{ e : E_p | \text{snds}(e,l) \neq \text{nul} \} \]

\[ \text{Rcv}_{p,l} = \{ e : E_p | \text{kind}(e) \text{ is receive on } l \} \]
Communication Diagram
Distributed Systems

A distributed system $D$ will be a graph $G$ whose nodes are processes and whose links are communication channels.

We will treat processes abstractly in these lectures. In our verification work we use message automata and IO Automata at the nodes. We can also imagine other abstractions – e.g. active objects – as well as CML or Java programs.
Causal Order

Following Lamport, we define the causal order, $e < e'$.

Let $e \sqsubseteq e'$ iff $\neg \text{first}(e)$ and $e = \text{pred}(e')$

or $e = \text{sender}(e')$

Take $e < e'$ as the transitive closure of $e \sqsubseteq e'$. 
Executions of Distributed Systems

Executions of distributed systems are event systems in a natural way. Executions are typically indexed by time, and that can be discrete, say \( t \in \mathbb{N} \).

At each moment of time, a process at \( i \) is in a state, \( s(i, t) \), and the links are lists of tagged messages, \( m(l, t) \). At each locus \( i \) and time \( t \), there is an action, \( a(i, t) \), taken. The action can be null, i.e., no state change, no receives, hence no sends.
Fair-Fifo Executions

We assume executions are fair: channels are loss-less; and fifo: messages are received in the order sent.

1. Only the process at $i$ can send messages on links originating at $i$.

2. A receive action at $i$ must be on a link whose destination is $i$ and whose message is at the head of the queue on that link.

3. There can be null actions that leave a state unchanged between $t$ and $t + 1$.

4. Every queue is examined infinitely often, and if it is nonempty, a message is delivered.

5. The precondition of every local action is examined infinitely often and if true the action is taken.
Event Systems of Fair-Fifo Executions

If $w$ is a fair-fifo execution of a distributed system $D$, we can define an event system from it, $\text{Ev}(w)$.

The \textbf{types} $\text{Loc}, \text{Lnk}, \text{Kind}, \text{Tag}, \text{Id}$ are inherited from $D$.

The \textbf{events} $E$ are the points $<i,t>$ locus $i$ and time $t$, at which a non-null action, local or receive, occurred in $w$. 
Specifying Protocols and Systems

Function specification:
\[ \vdash ?x : A. \, ?y : B. \, R(x, y) \text{ ext } f : A \to B \]

Protocol specification is part of a system specification.
\[ \overline{x} : \text{System} \vdash \text{ProcessSpec}(x_i) \text{ ext } P_i \]

We will focus on the ProcessSpec.
Axioms

**Axiom 1:** For every event \( e \) which sends a list of messages on link \( l \), we can find an event \( e' \) at the destination of \( l \) at which all messages sent are received.

\[
\text{receive}(e '') \Rightarrow e = \text{sender}(e '') \\
\text{link}(e '') = l \Rightarrow (e ' = e '' \lor \text{loc}(e ') = \text{dst}(l))
\]

**Axiom 2:** The predecessor function at each locus is one-to-one.

\[
? e , e ' : E . \ \text{loc}(e) = \text{loc}(e ') \Rightarrow \\
(\text{pred}(e) = \text{pred}(e ') \Rightarrow e = e ')
\]
Axioms

Axiom 3: The causal order $<$ is strongly well-founded.

$$\exists f : E \rightarrow \mathbb{N}. \ \forall e, e' : E. \ (e < e' \Rightarrow f(e) < f(e'))$$

Axiom 4: If $e$ is not the first event at a locus, then its predecessor is at the same location.

$$? e : E. \ \neg \text{first}(e) \Rightarrow \text{loc}(\text{pred}(e)) = \text{loc}(e).$$

Axiom 5: If $e$ is a receive event, then the locus of the sender is the source of the link of $e$.

$$? e : E. \ \text{receive}(e) \Rightarrow \text{loc}(\text{sender}(e)) = \text{src}(\text{link}(e)).$$
**Axioms**

**Axiom 6:** Messages on a link are received in the order sent.

\[ \forall e, e': E \cdot (rcv(e) \Rightarrow rcv(e')) \Rightarrow \]
\[ (link(e) = link(e')) \Rightarrow \]
\[ (sender(e) < sender(e')) \Rightarrow e < e'. \]

**Axiom 7:** By convention, for any event except the first, to say that an observable \( x \) has a value \( v \) when event \( e \) happens at the locus of \( e \) is to say that \( x \) after the predecessor of \( e \) is \( v \).

\[ \forall e : E \cdot \neg first(e) \Rightarrow \exists x : Id \cdot \]
\[ (x \text{ when } e = x \text{ after } \text{pred}(e)) \]
Deriving Algorithms and Protocols

Sequential program derivation from a specification:

\[ \vdash \exists p : \text{State} \rightarrow \text{State}. \ ?s : \text{State}. \ R\ (s, p\ (s)) \]

Nuprl supports extraction.

\[ \vdash \ ?s : \text{State}. \ ?s' : \text{State}. \ R\ (s, s') \text{ ext} \ p \]
Refinements for Programs

\[ \vdash \ ? x : A. ? y : B. R(x, y) \text{ ext } \lambda x. \text{cut}(x, z.g(x, z); l(x)) \]

\[ x : A \vdash ? y : B. R(x, y) \text{ ext } \text{cut}(x, z.g(x, z); l(x)) \text{ by cut L} \]

1. \[ x : A, z : L \vdash ? y : B. R(x, y) \text{ ext } g(x, z) \text{ by Do } g(x, z) \%
   \]

\[ x : A, z : L \vdash R(x, g(x, z)) \]

2. \[ x : A \vdash L \text{ ext } l(x) \text{ by } \_\_\_ \_\_\_ \]

\[ x : A \vdash R(x, g(x, z)) \text{ by Do } g(x, z) \%
\]
Refinements for Systems

\[
\vdash \ ?D : \text{System. } ?\ es : \text{ES} (D).
\]

\[
R (es) \ \text{ext} \ \text{Comp} \ (pf_1 (D_1, es_1), pf_2 (D_2, es_2))
\]

by Comp

1. \(D_1 : \text{System} (G, \text{Loc}, \text{Llk})\)
   \(es_1 : \text{ES} (D_1) \vdash R_1 (es_1) \ \text{ext} \ pf_1 (D_1, es_1)\)

2. \(D_2 : \text{System} (G, \text{Loc}, \text{Llk})\)
   \(es_2 : \text{ES} (D_2) \vdash R_2 (es_2) \ \text{ext} \ pf_2 (D_2, es_2)\)
Event Systems $es$ of Distributed System $D$

For an (abstract) distributed system $D$, say that $es$ is an event system of $D$, $es \in ES(D)$, if $es$ is the event system $Ev(w)$ of an execution $w$ of $D$. 
Deriving the Two-Phase Handshake

We illustrate this process by deriving a protocol for the two-phase handshake from a proof that its specification is realizable.

(1) $\forall e_1, e_2 : \text{Snd}_{S, l_1} \cdot \ ? r : \text{Rcv}_{S, l_2} \cdot (e_1 < e_2 \Rightarrow e_1 < r < e_2)$

(2) $\forall e_1, e_2 : \text{Snd}_{R, l_2} \cdot \ ? r_1, r_2 : \text{Rcv}_{R, l_1} \cdot (e_1 < e_2 \Rightarrow r_1 < e_1 < r_2 < e_2)$
Deriving Handshake - 1

One way to achieve alternation of sends and receives at $S$ is to introduce a boolean variable $rdy$ and stipulate that $S$ sends only when $rdy$ is true. When a send on $l_1$ occurs, $rdy$ is set to false.

**Lemma 1:** For any two processes $S, R$, and links $l_1, l_2$,

\[
\text{src}(l_1) = S \quad \& \quad \text{dst}(l_1) = R, \text{src}(l_2) = R \\
\& \quad \text{dst}(l_2) = S, \quad \& \quad rdy \text{ in } l_1,
\]

we can realize an event system with the property that initially $rdy = \text{true}$ at $S$.

Call the realizer for Lemma 1 $ES_1$. 
Deriving Handshake - 2

Lemma 2: We can find a realizer that extends \( ES_1 \) such that for any non-empty type \( T \),

\[
\forall e : E_S. (\text{rdy when } \, e = \text{true}) \implies \\
\exists e' : E_S. e < e' \land (\exists v : T. (\text{rdy after } \, e' = \text{false} \land \\
\text{sends (e',l) = [l_1,val,v]} \lor \text{rdy when } \, e' = \text{false}).
\]

Proof:

Infinitely often the process at \( S \) will examine its precondition \( \text{rdy = true} \). If \( \text{rdy} \) has not changed, then the send will occur, and \( \text{rdy} \) will be set to false. Otherwise, \( \text{rdy} \) must be false at \( \, t \), so some event \( \, e' \) after \( \, e \) set it to false.

Qed

Call the realizer for Lemma 2 \( ES_2 \).
Deriving Handshake - 3

Lemma 3: We can find a realizer that extends \( ES_2 \) such that

\[
\forall e : E_S. \text{ sends } (e, l_1) \neq \text{nil} \implies rdy \text{ after } e = \text{false}.
\]

Proof:

We constrain the realizer of Lemma 2 to satisfy the frame condition that there are no actions that send on \( l_1 \) with tag \( \text{val} \) except for the one specified in Lemma 2.

Qed

Call the realizer for Lemma 3 \( ES_3 \).
Deriving Handshake - 3

Lemma 4: We can find a realizer that extends $ES_3$ such that

$$\forall e : E_S. \ rcv(e, l_2) \Rightarrow rdy \ after \ e = true.$$  

Proof:

We add to the process at $S$ a response to any receive action on $l_2$, namely this receive will set $rdy$ to true.

Qed

Call this realizer $ES_4$. 
Deriving Handshake - 4

**Lemma 5**: We can find realizers that extend $ES_5$ such that

$$\forall e : E \_S \cdot \text{rcv}(e, l_2) \Rightarrow ? e' : E \_S \cdot e < e' \land \text{sends}(e, l_1) \neq \text{nil}.$$ 

**Proof**: 

We add the frame condition that only a receive on $l_2$ or reset of $\text{rdy}$ to false can affect $\text{rdy}$.

Note, $\text{rdy}$ after $e=\text{true}$ by Lemma 4. Execution is fair, thus the precondition of the send will be checked infinitely often, hence after $e$. Only a reset can change $\text{rdy}$ to false by the frame condition, and this happens only if the send on $l_1$ is executed. Thus there will be an event $e'$ after $e$ such that the send is executed.

Qed

Call this realizer $ES_5$. 
Theorem 1: Any realizer extending $ES_5$ satisfies

$$\forall e_1, e_2 : Snd_{S,l_1}. (e_1 < e_2 \Rightarrow \exists r : Rcv_{S,l_2}. e_1 < r < e_2)$$

Proof:

Let $e_1, e_2$ be send events at $S$ on link $l_1$. Suppose $e_1 < e_2$. By Lemma 3, $rdy$ after $e_1 = \text{false}$. The only event that can set $rdy$ to true is a receive event. The only way $e_1 < e_2$ is possible is that $rdy$ when $e_2 = \text{true}$. Thus before $e_2$ and after $e_1$ there must be a receive event.

Qed
Lamport’s TLA+ Specification - 2003

EXTENDS Naturals CONSTANT Data
VARIABLES val, rdy, ack
TypeInvariant \( \triangleq \) \( \wedge \text{val} \wedge \text{Data} \wedge \text{rdy} \wedge \{0,1\} \wedge \text{ack} \wedge \{0,1\} \)

Init \( \triangleq \) val \wedge Data \wedge rdy \wedge \{0,1\} \wedge \text{ack} = \text{rdy}
Send \( \triangleq \) \( \wedge \text{rdy} = \text{ack} \)
\( \wedge \text{val}' \wedge \text{Data} \)
\( \wedge \text{rdy}' = 1 - \text{rdy} \)
\( \wedge \text{UNCHANGED} \ \text{ack} \)
Rcv \( \triangleq \) \( \wedge \text{rdy} \neq \text{ack} \)
\( \wedge \text{ack}' = 1 - \text{ack} \)
\( \wedge \text{UNCHANGED} < \text{val}, \text{rdy} > \)
Next \( \triangleq \) Send \lor Rcv
Spec \( \triangleq \) Init \land \Box [\text{Next}] <val, rdy, ack>

THEOREM Spec \( \Rightarrow \Box[\text{TypeInvariant}] \)
# Types and Sets

<table>
<thead>
<tr>
<th>Types</th>
<th>Sets</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\pi$</td>
<td>$\pi$</td>
</tr>
<tr>
<td>$\times$</td>
<td>$\times$</td>
</tr>
<tr>
<td>$\top$</td>
<td>$\oplus$</td>
</tr>
<tr>
<td>$\Sigma$</td>
<td>$\Sigma$</td>
</tr>
</tbody>
</table>