ENSE 623 Systems Validation and Verification

Model-Based Design with LTSA

Mark Austin

E-mail: austin@isr.umd.edu

Institute for Systems Research, University of Maryland, College Park

Table of Contents

Model-Based Design with LTSA

The labeled transistion system analyzer (LTSA) is a tool for:

> **...validating communication and sequencing among entities in systems containing concurrent behaviors. LTSA mechanically checks that the specification of a concurrent system satisfies the properties required of its behavior.**

In LTSA ...

> **... processes correspond to sequences of actions.**

The power of LTSA lies in its ability to link processes through

> **... shared actions and compose processes side by side to create systems running concurrently on many levels.**

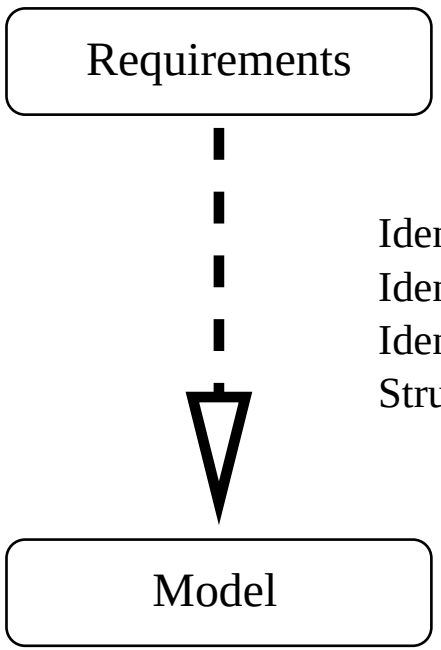
Spatial/temporal design is deliberately abstracted from from modeling consideration.

The textual representation is the finite state process (FSP) language. Labeled transition systems (LTSs) are the graphical representation.

This tool runs as an applet on JAVA Runtime Environment 1.3 or higher.

Model-Based Design with LTSA

Pathway from Requirements to Architecture-Level Design



Goals of the system...
Scenarios / Use case models....
Properties of interest ....

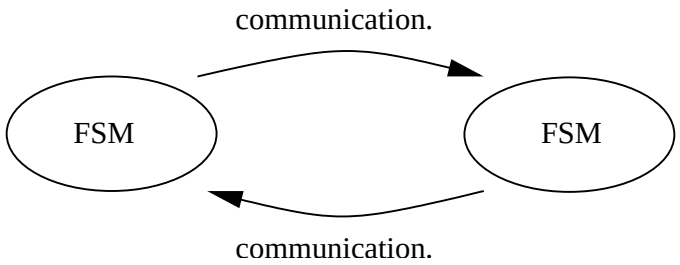
Identify main events, actions, and interactions ....
Identify and define main processes ...
Identify and define properties of interest ...
Structure processes into an architecture

Check traces of interest ...
Check properties of interest ....

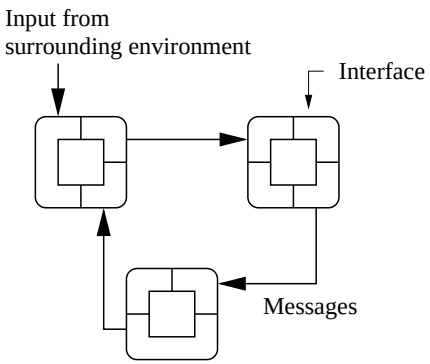
A top-down specification of required behavior for components can be specified through the use of visual modeling languages such as UML.

Behavior Modeling with LTSA

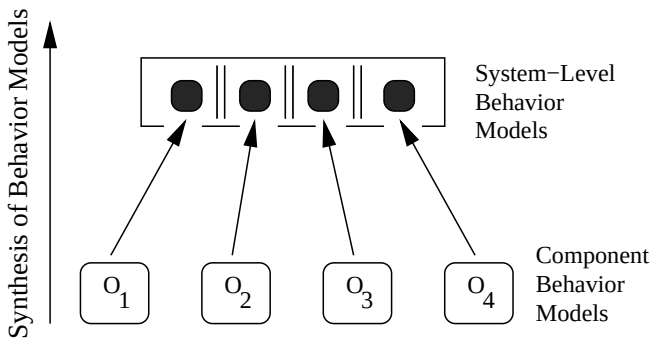
Systems are modeled as networks of interacting finite state machines



Models of system-level behavior are synthesized from component- and subsystem-level behaviors.



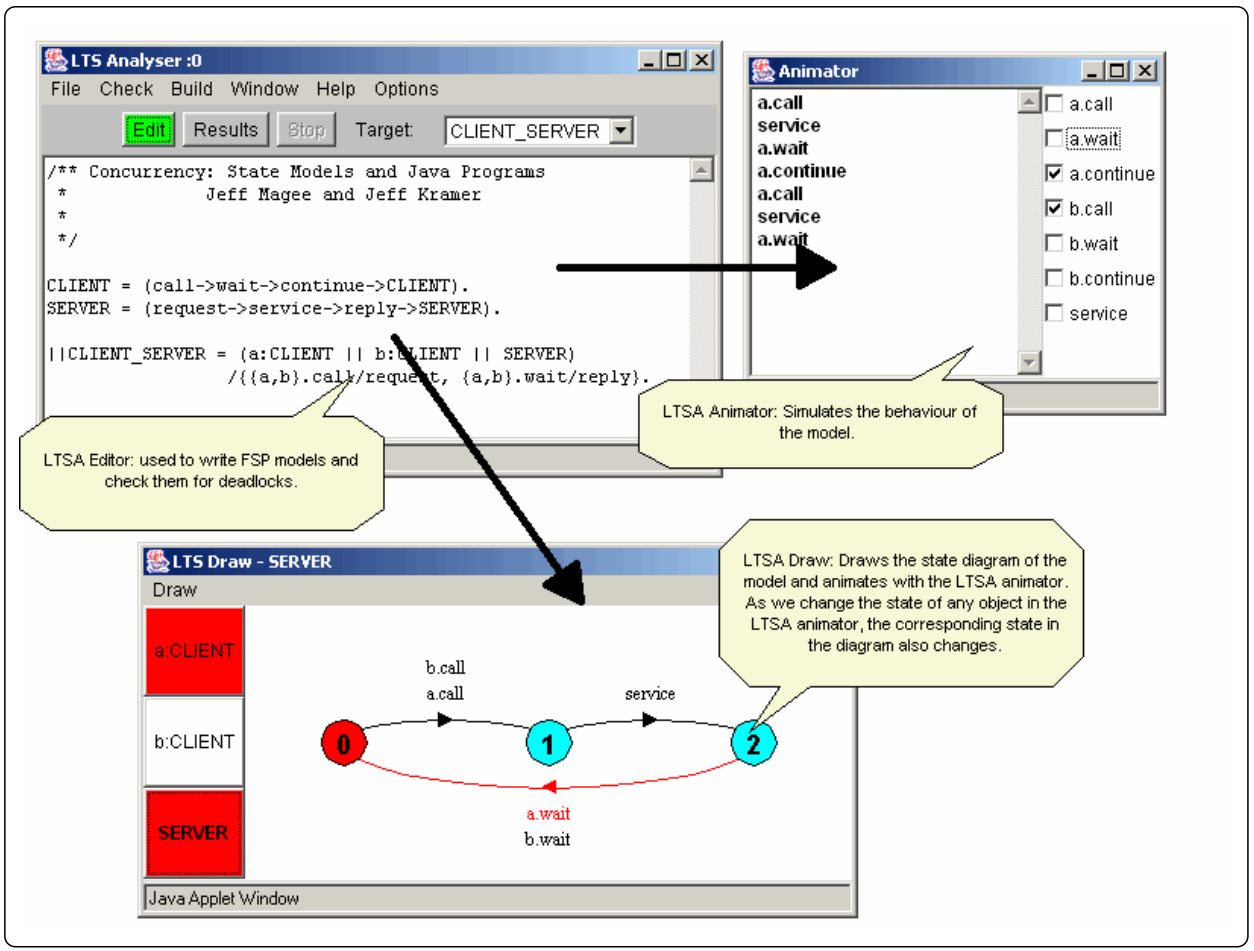
Systems correspond to a network of communicating objects.



Synthesis of System−Level Behavior Models from Component−Level Behavior Models

Behavior Modeling with LTSA

Schematic of components in LTSA



Behavior Modeling with LTSA

The LTSA Components

1. **LTSA Editor**

   The LTSA editor is used to write FSP models and check them for deadlocks.

2. **LTSA Draw**

   Draws the state diagram of the model and animates with LTSA LTSA animator.

   When the state of an object changes in the LTSA animator, the corresponding state in the LTS Draw diagram also changes.

3. **LTSA Animator**

   Simulates behavior of the model/system by stepping through the system actions and events.

A node corresponds to a possible state in the system. Edges correspond to a change of state in the system due to an event.

Finite State Processing Language

The Finite State Processing Language (FSP) contains:

Actions (including shared and guarded actions),

- Identify with a name each possible event of importance.
- If concurrent actions have common elements, then there will be interleaving of their behaviors linked through common actions.

Traces (and sets of Traces)

- System execution corresponds to a step-by-step sequence of actions.
- Some systems will execute different ways on different occastions – therefore, all traces of actions are important.

To understand how a process executes we can build ...

... a state machine representation of the FSP execution.

Finite State Processing Language

FSP also supports:

Parallel Composition of Processes

- Given two labeled transition systems (LTSs) P_1 and P_2, we denote the parallel composition $P_1 || P_2$ as the LTS that synchronizes actions common to both processes and interleaves the remaining actions.
- By extension the architectural-level behavior model is defined by:

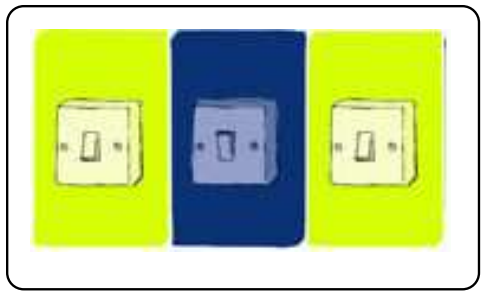
$$\text{Architecture-Level Behavior Model} = P_1 || P_2 || P_3 \cdots P_n$$

  where P_i is the finite state model for the i-th component among n interacting components.
- Joint behavior is the result of all LTSs executing asynchronously, but synchronizing on all shared message labels.
- At the architecture level, labeled transition system nodes represent system-level states, which, in turn, correspond to specific combinations of component-level states.

Behavior Modeling with LTSA

Example 1. Behavior of a Three-Way Switch

Consider the concurrent behavior of three switches working together.



The FSP model for behavior of a single switch is:

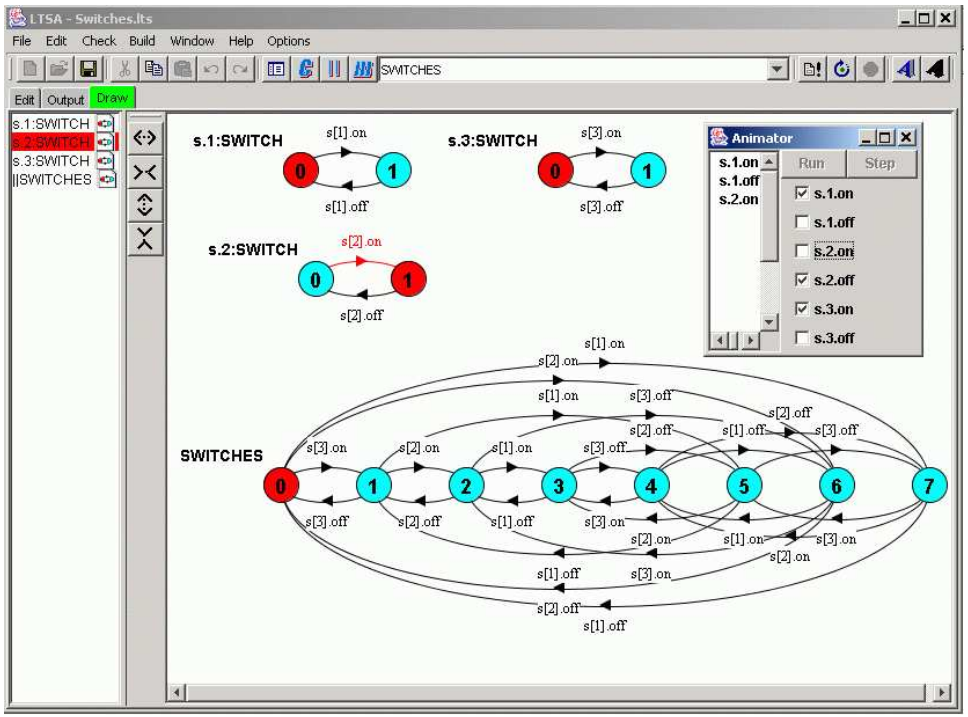
```
SWITCH = (on->off->SWITCH).
```

Models for two and three switches:

```
|| TWO_SWITCH = (a:SWITCH || b:SWITCH).
|| SWITCHES(N=3) = (forall[i:1..N] s[i]:SWITCH).
```

Behavior Modeling with LTSA

Example 1. Animator Box for Behavior of a Three-Way Switch



For each switch there is one LTS, labeled:

`s.1:SWITCH, s.2:SWITCH s.3:SWITCH.`

Behavior Modeling with LTSA

Example 2. Modeling Behavior of a Door and Handle

In the fragment of LTS code:

```
HANDLE      = ( down -> up    -> HANDLE).
DOOR        = ( open -> close -> DOOR ).
```

defines processes for a handle and a door, each having behavior defined by transitions between two states.

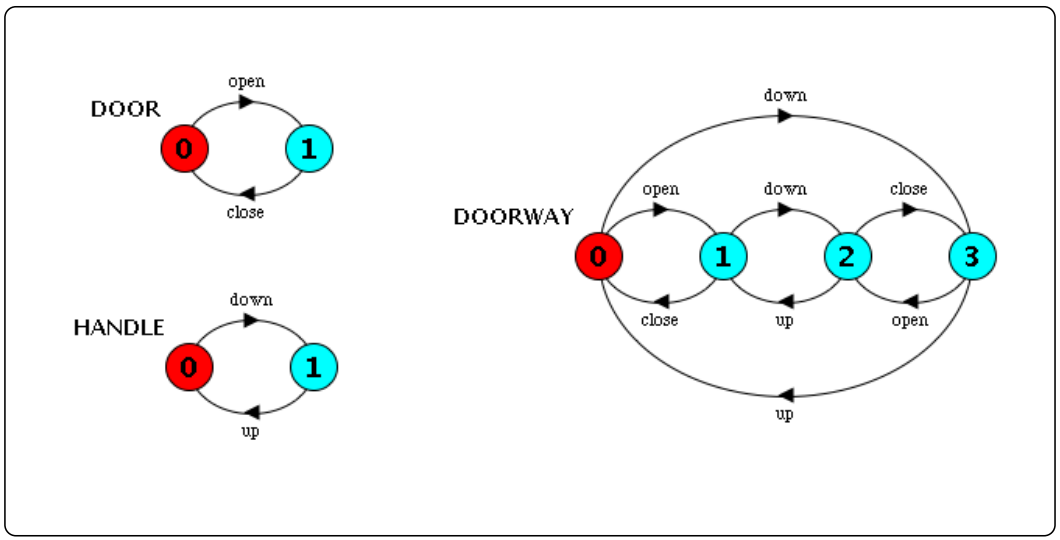
- The alphabet for the HANDLE process is $(down, up)$.
- The DOOR process has an alphabet $(open, close)$.
- Accordingly, for the handle and door, transitions are `up` and `down` and `open` and `close`, respectively.
- Behavior of the handle is constrained to follow the action sequence: `down -> up -> down -> up -> down` ..... and so on.

Behavior Modeling with LTSA

Example 2. Composition of the Door and Handle Processes

```
||DOORWAY = ( HANDLE || DOOR ).
```

Results in:



Behavior Modeling with LTSA

Example 2. Interpretation of Composed Behavior

The doorway process moves among four states having the following interpretation:

Component behaviors		Composed behavior	
Handle System	Door System	System Tuple	Simplified Notation
up	close	(up, close)	State 0
up	open	(up, open)	State 1
down	open	(down, open)	State 2
down	close	(down, close)	State 3

Note. Because the alphabet of actions for door (i.e., `open`, `close`) and handle (i.e., `up`, `down`) are disjoint, the door and handle processes can advance in any order, subject to the action-sequence constraints within each process.

Behavior Modeling with LTSA

Simultaneous Execution of Shared Actions

When action names in a process composition are common, these actions are said to be shared.

LTSA uses shared actions as ...

> **... the mechanism to synchronize interactions among processes.**

While the shared actions may be arbitrarily interleaved,

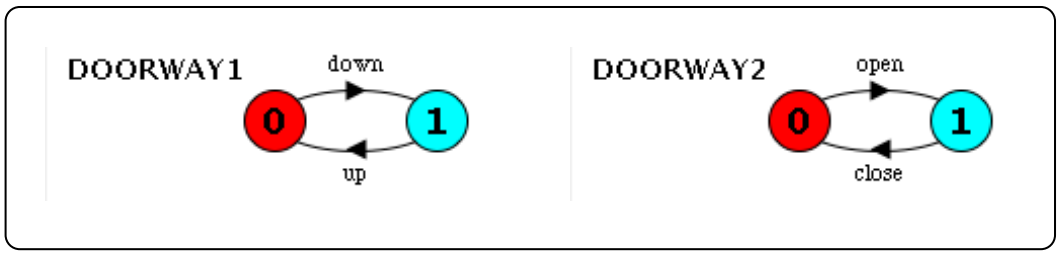
> **... the shared actions must be executed simultaneously by all participating processes.**

Behavior Modeling with LTSA

Example 3. Doorway Process Model with Action Relabeling and Shared Actions

Consider the script of FSP code and corresponding LTS:

```
||DOORWAY1 = ( HANDLE || DOOR )/{down/open,up/close}.
||DOORWAY2 = ( HANDLE || DOOR )/{open/down,close/up}.
```



Points to note:

- In `DOORWAY1` the action names `close` and `open` are changed to `up` and `down`, respectively.
- In `DOORWAY2` the action name are switched.

Behavior Modeling with LTSA

Deterministic Choice

- If “x” and “y” are actions, then

(x->P | y->Q)

describes a process which initially engages in either the actions x or y.

The execution of action x will have subsequent behavior described by P. Similarly, the execution of y will have subsequent behavior described by Q.

Non-Deterministic Choice

- The process

(x->P | x->Q)

is non-deterministic because after the action x, behavior may be described by either process P or process Q.

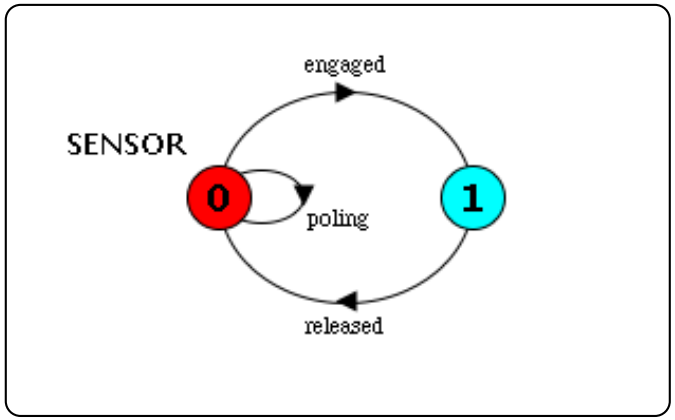
Behavior Modeling with LTSA

Example 4. Sensor Model that involves Deterministic Choice

This scenario can be easily implemented via,

```
SENSOR      = (   engaged -> released -> SENSOR
              |   poling -> SENSOR ).
```

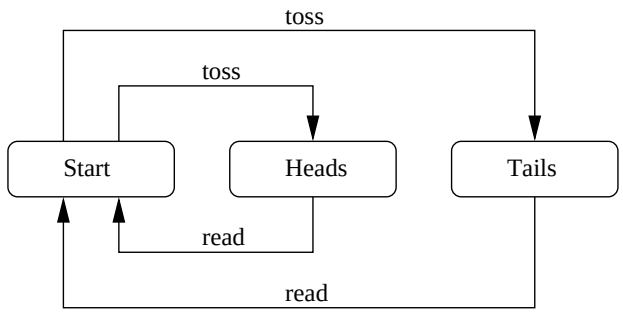
LTS for Sensor Behavior



Behavior Modeling with LTSA

Example 4. Toin Tossing Model that involves Non-Deterministic Choice

Scenario ... Behavior Model



Set of states = $\{Start, Heads, Tails\}$; Set of actions = $\{toss, read\}$.

FSP code:

```
START = (  toss -> HEAD -> read -> START
         | toss -> TAIL -> read -> START ).
```

Behavior Modeling with LTSA

Definition of Tagged Processes

Individual processes can be tagged, thereby providing a mechanism ...

> **... for more than one copy of a process to be used and uniquely identified in a system model.**

The notation a:P prefixes each action label in the alphabet of process P with the label a.

Example 6. Process Behavior in a Two-Sensor System

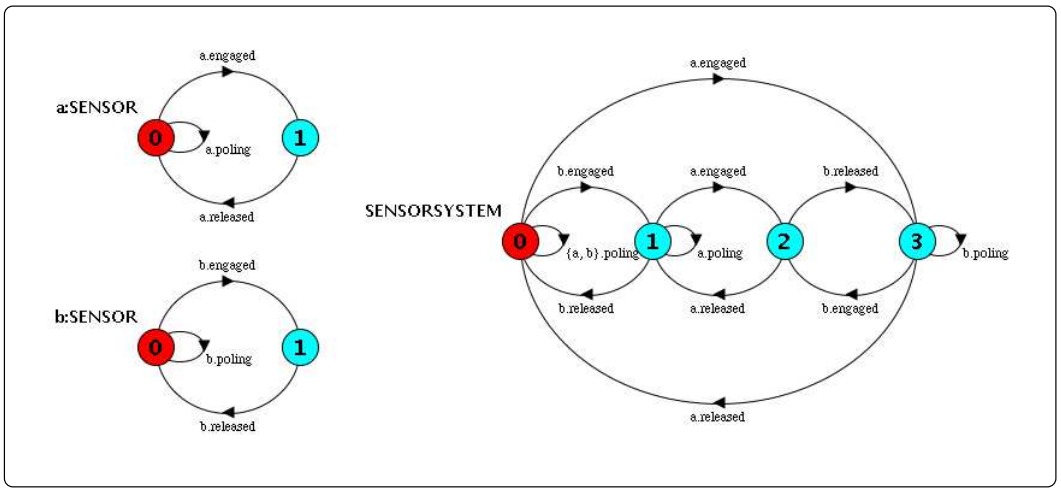
The fragment of code:

```
SENSOR      = (   engaged -> released -> SENSOR
                | poling -> SENSOR ).

||SENSORSYSTEM = ( a:SENSOR || b:SENSOR ).
```

creates a composite behavior model from two individual sensor behaviors (i.e., a:SENSOR and b:SENSOR).

Behavior Modeling with LTSA

LTS and Composite State Interpretation



Sensor System	a:SENSOR	b:SENSOR	Interpretation
Composite state 0	state 0	state 0	a and b are poling
Composite state 1	state 0	state 1	a is poling, b is engaged
Composite state 2	state 1	state 1	a and b are both engaged
Composite state 3	state 1	state 0	a is engaged, b is poling

Behavior Modeling with LTSA

Guarded Actions

An action that is conditional on a particular condition being true is terms a guarded action.

The FSP syntax for guarded actions is

(when B x->P | y->Q)

When guard B is true, actions x and y are both eligible to be chosen. Otherwise, only action y can be chosen.

Behavior Modeling with LTSA

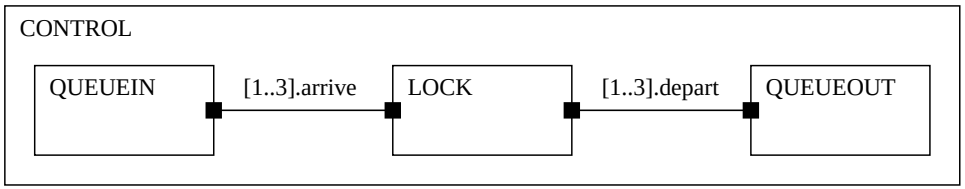
Example. Convoy of Three Ships Traversing a Lock System

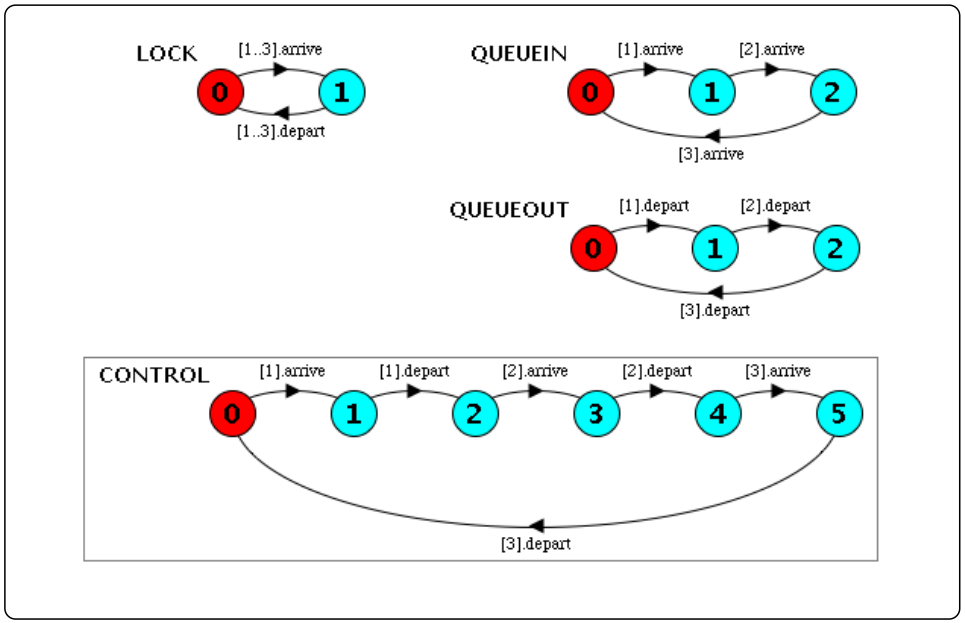
```
const N  =   3  // Number of ships passing through lock ...
const M  =   4  // Number of states in holding pattern queue...
range IQ =1..M  // Queue count

// Define input/output queues

   QUEUEIN = QUEUEIN[1],
   QUEUEIN[i:IQ]  = ( when (i<M) [i].arrive -> QUEUEIN[i%N+1] ).
   QUEUEOUT = QUEUEOUT[1],
   QUEUEOUT[i:IQ] = ( when (i<M) [i].depart -> QUEUEOUT[i%N+1] ).

// Model spaces in lock ....

   LOCK = SPACES[1],
   SPACES[i:0..1] = ( when(i>0) [j:1..N].arrive  -> SPACES[i-1]
                    | when(i<1) [j:1..N].depart  -> SPACES[i+1] ).

// Create model of controled lock system behavior ...

   ||CONTROL = ( LOCK || QUEUEIN || QUEUEOUT ).
```

Behavior Modeling with LTSA

Structure Diagram and LTS

The lock is a bounded buffer of maximum capacity one.





Behavior Modeling with LTSA

Modeling of Shared Resources

In the previous example, ...

... orderly use of the lock system was enforced through the use of queue processes.

A second possibility is to ...

... to define independent ship processes, and then simply state that the lock system process is a shared resource that can only be used by one ship process at a time.

With FSP, this is achieved with ...

... the double-colon syntax ::, as in $\{a_1, a_2, .. a_y\}$::P.

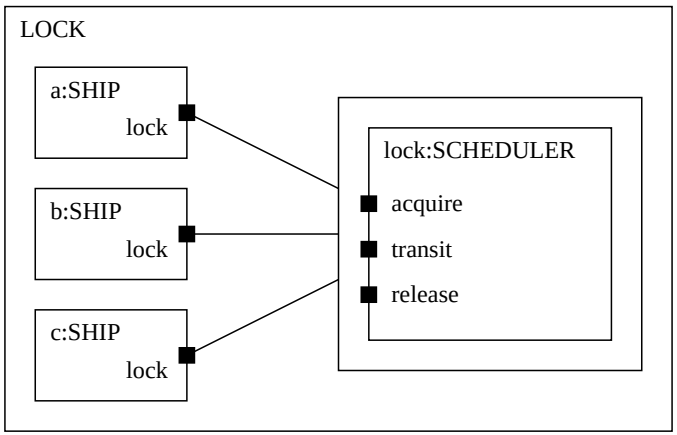
The latter replaces every label n in the alphabet of P with labels a_1.n, a_2.n through a_y.n.

Behavior Modeling with LTSA

Example. Lock Scheduler Modeled as a Shared Resource

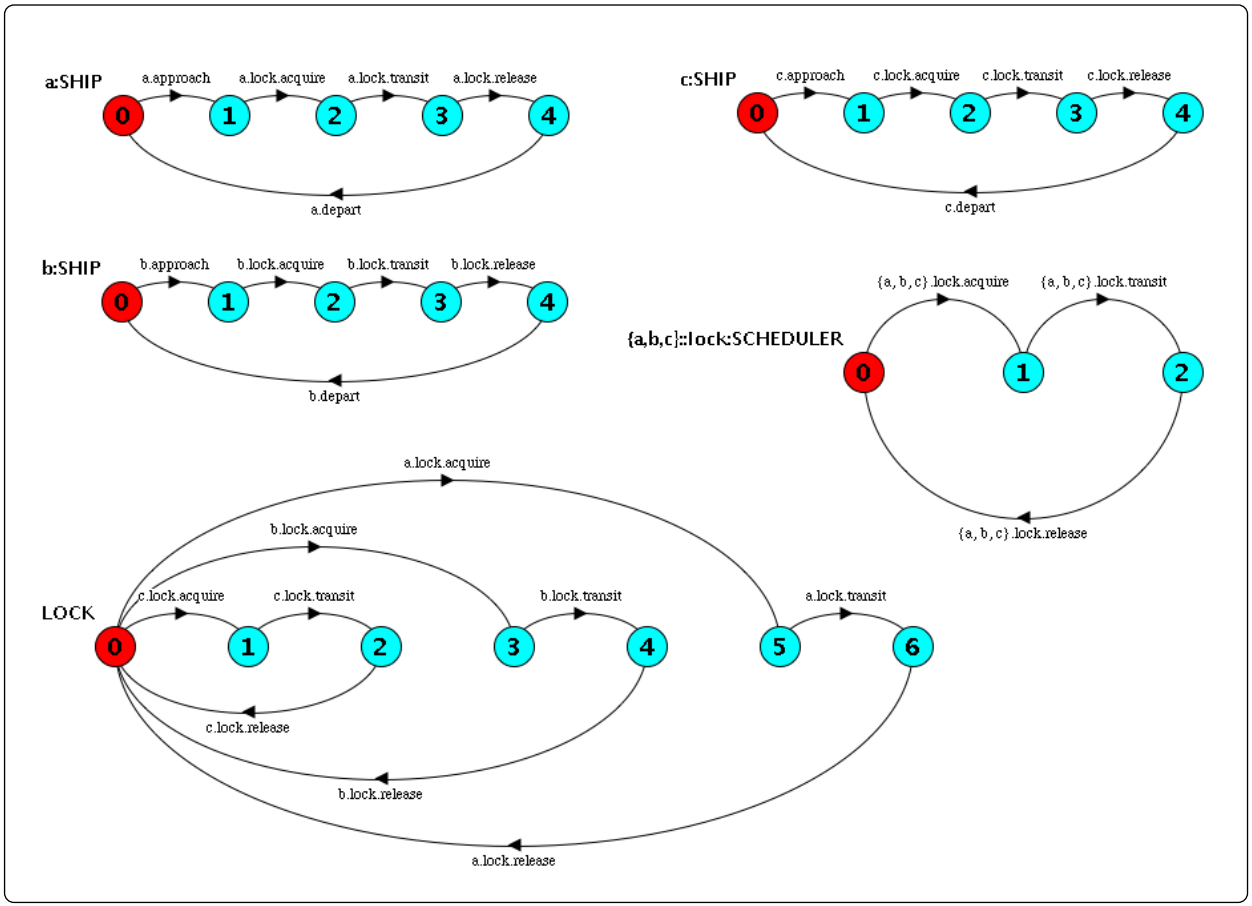
```
// Define ship and scheduler processes, then compose behavior model..

   SHIP      = ( approach -> lock.acquire -> lock.transit -> lock.release ->
                 depart -> SHIP ).
   SCHEDULER = ( acquire -> transit -> release -> SCHEDULER ).

 ||LOCK = ( a:SHIP || b:SHIP || c:SHIP || {a,b,c}::lock:SCHEDULER ) \
          {a.approach,a.depart,b.approach,b.depart,c.approach,c.depart}.
```

Structure Diagram for Lock Scheduler



Behavior Modeling with LTSA

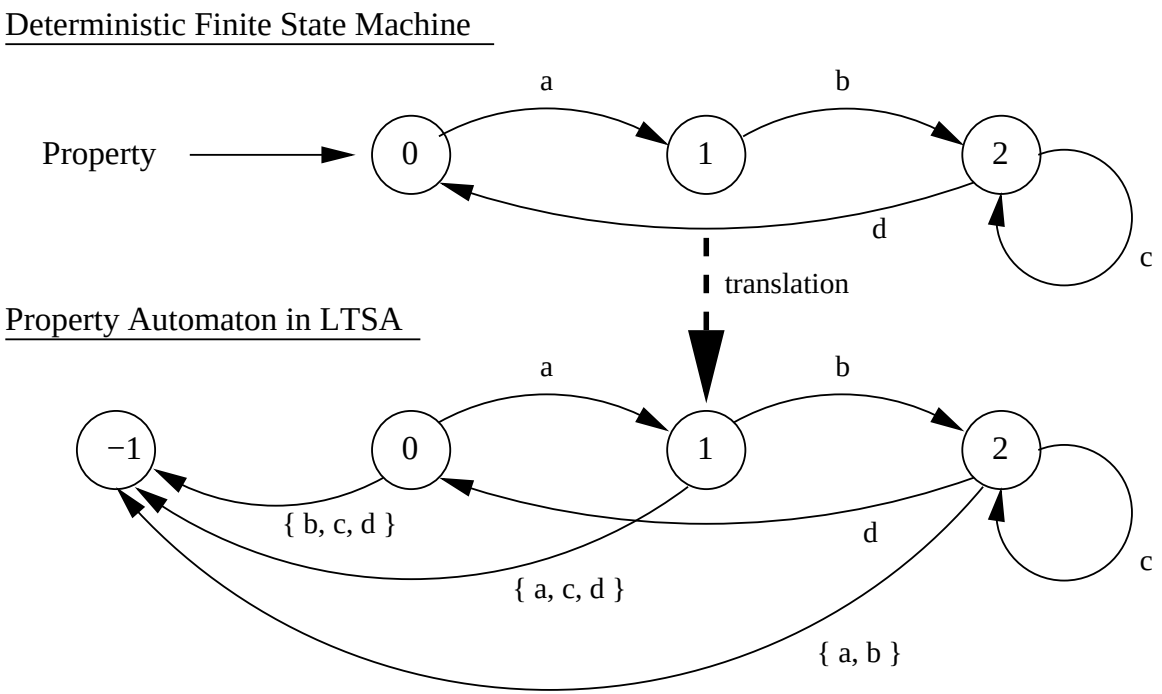
LTS for Lock Scheduler



Model Checking in LTSA

Compositional Approach to Model Checking

Procedure for definition of property automata in LTSA ...



Model Checking in LTSA

Definition of Safety Properties

Safety properties are specified in FSP by property processes (a.k.a., determinstic finite-state machines called property automata).

> **A safety property ”P” defines a deterministic process that asserts any trace, including actions in the alphabet of P, will be accepted by P.**

Thus, if "P" is composed with "S" then the trace of actions in the set:

> **P intersction S**

must also be valid traces of P.

Otherwise an error occurs.

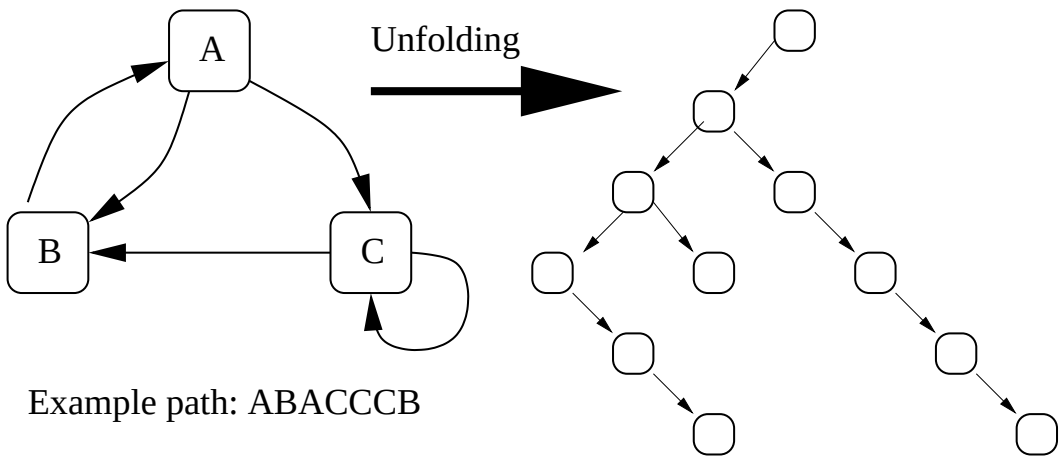
Any deviation from this order results in a safety error.

Model Checking in LTSA

Procedure for Evaluation of Safety Properties

Visual representation of system C composed from processes A and B, and, validation of system C via composition with property automata.

Evaluation of system properties through fsm unfolding and exhaustive search ....



Example path: ABACCCB

Model Checking in LTSA

Example 9. Formal Validation for a Model of Polite Conversation

```
// ================================================================================
// Jack and Diane have conversation over coffee ....
// ================================================================================

// Create a person who: (1) talks and drinks coffee, or
//                      (2) just waits and then drinks coffee ....

   PERSON = (   talk -> drink -> PERSON
              |  wait -> drink -> PERSON ).

// Jack and Diane meet ....

   minimal ||JACK_AND_DIANE_MEET = ( jack:PERSON || diane:PERSON ).

// To learn, conversation needs to be two way ....

   TWO_WAY = ( jack.talk -> diane.talk -> TWO_WAY ).
```

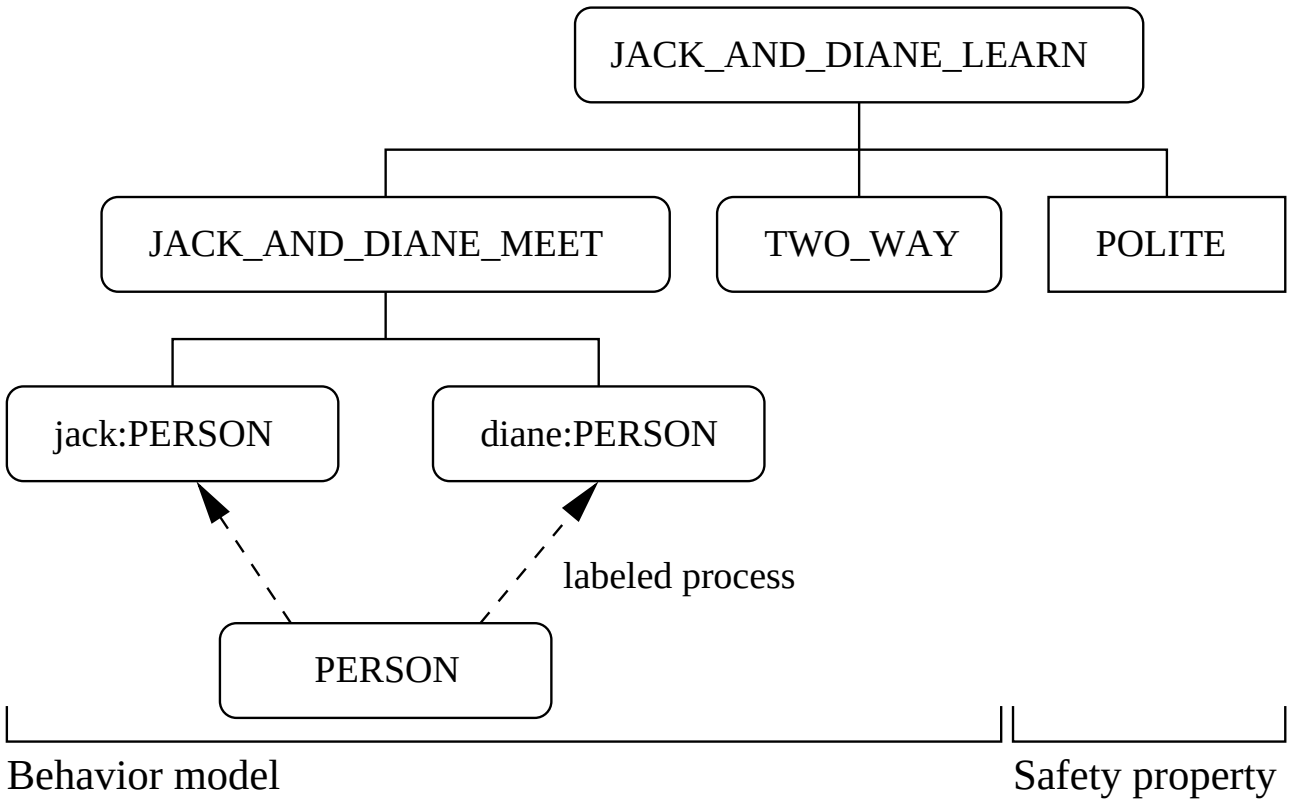
Model Checking in LTSA

Example 9. FSP code continued

```
// Define a property for polite conversation ...

   property POLITE = ( jack.talk -> diane.talk -> POLITE ).

// Check that the conversation model is in fact polite ...

   minimal ||JACK_AND_DIANE_LEARN = ( JACK_AND_DIANE_MEET || TWO_WAY || POLITE ) / {
                                     jack.talk/diane.wait, diane.talk/jack.wait }.

// Check progress properties

   progress DIANE_TALKS = { diane.talk }
   progress JACK_TALKS  = {  jack.talk }

// ================================================================================
// End!
```

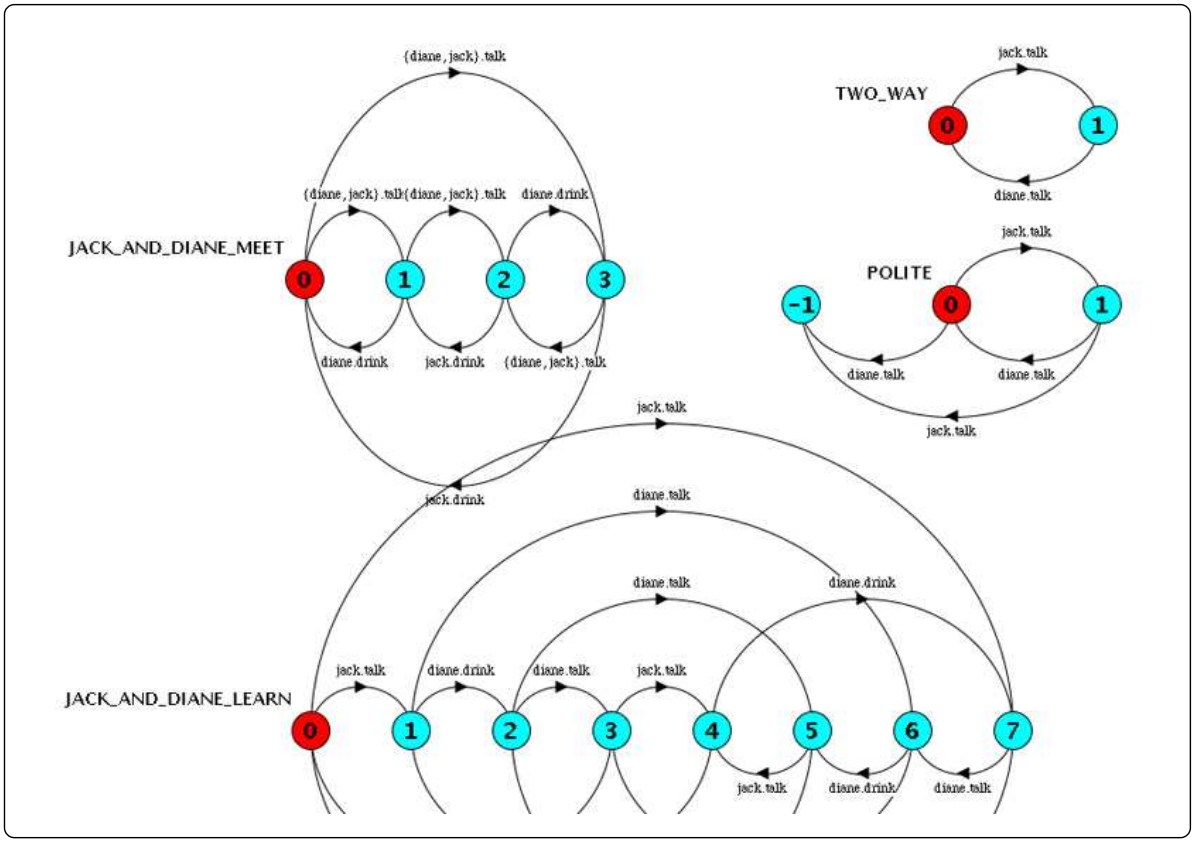
Model Checking in LTSA

Example 9. Process Hierarchy for Behavior Model



Model Checking in LTSA

Example 9. Behavior Models for Polite Conversation



Model Checking in LTSA

Example 9. How do we know that the model checking worked?

Notice that POLITE will ...

> **... transition to an error state if Diane talks more than once or, alternatively, Jack talks more than once.**

Key Point. If the composed process

```
( JACK_AND_DIANE_MEET || TWO_WAY || POLITE )
```

contains any of these sequences, then ...

> **... it too will also have an error state indicating that our model of behavior is not polite.**

The progress checks generate

```
Progress Check...
-- States: 8 Transitions: 16 Memory used: 1951K
No progress violations detected.
Progress Check in: 40ms
```

Case Studies and Follow-Up

Simplified Airpsace Management Tool

- See pages 331-346 of the class notes.

Behavior Modeling for Ships Passing through the Panama Canal

- See pages 347-392 of the class notes.

Follow-Up: Linking LTSA to Animation

- Magee J., et al, Graphical Animation of Behavior Models, ICSE 2000, Limerick, Ireland, 2000.

Slides from the LTSA Book

- The supplementary material contains a comprehensive set of slides from Jeff Kramer and Jeff Magee's book on LTSA, Concurrency: State Models and Java Programs (2nd Edition), John-Wiley and Sons, 2006.

References

- Magee J.L. Kramer J., Uchitel S., Labeled Transition System Analyzer (LTSA) Home Page, See:http://www.doc.ic.ac.uk/ jnm/book/ltsa/LTSA.html, 2004.
- Magee J.L., and Kramer J., Concurrency: State Models and Java Programs (2nd Edition), John Wiley and Sons, New York, 2006.
- Thorton K., A Simplified Airspace Management Tool, ENSE 622/ENSE 623 Project, Spring and Fall Semesters, 2005.