Software Verification and Testing

Lecture Notes: Transition Systems II

LTSs for Concurrent Systems

so far: we have modelled single sequential action systems

task: model concurrent systems, i.e.,

- model their parallel temporal development
- model their interaction/communication via shared variables or message passing

analogy:

- in engineering, temporal system behaviour is modelled by system of differential equations
- states correspond to vectors
- interaction is modelled by shared variables

here: systems are discrete, not continuous. . .

LTSs for Concurrent Systems

definition: for $i = 1, \ldots, n$ let $\mathcal{A}_i = (S_i, T_i, \alpha_i, \beta_i, \lambda_i)$ be LTSs.
Their free product $\mathcal{A}_1 \times \cdots \times \mathcal{A}_n$ is the LTS $\mathcal{A} = (S, T, \alpha, \beta, \lambda)$ defined by

- $S = S_1 \times \cdots \times S_n$
- $T = T_1 \times \cdots \times T_n$
- $\alpha(t_1, \ldots, t_n) = (\alpha(t_1), \ldots, \alpha(t_n))$
- $\beta(t_1, \ldots, t_n) = (\beta(t_1), \ldots, \beta(t_n))$
- $\lambda(t_1, \ldots, t_n) = (\lambda(t_1), \ldots, \lambda(t_n))$

remarks:

- global states of a concurrent systems are now vectors
- global transitions transform state vectors into state vectors
- this makes actions synchronous, i.e., a clock drives the individual transitions

LTSs for Concurrent Systems

observation: in communicating systems, many global transitions cannot be carried out due to synchronisation constraints

example: consider the boolean variable and the sequential program example

- the boolean variable has 2 states and 8 transitions
- the sequential program has 4 states and 5 transitions
- the free product has 8 states and 40 transitions (combinatorial explosion. . .)
- however, in the combined system, the sequential program sets the states of the boolean variable

LTSs for Concurrent Systems

example: (cont.)

- so there is only one global transition

$$(4, true) \rightarrow_{(b:=false,b:=false)} (1, false)$$

  from global state $(4, true)$
- similarly, the test of b in the sequential program is always synchronised with a read action on the boolean variable

notation: we will write $true!$, $true?$ etc. for send and receive actions

LTSs for Concurrent Systems

example: (cont.)

- legal global actions:

$$(b := true, b := true)$$

$$(b := false, b := false)$$

$$(true?, true!)$$

$$(false?, false!)$$

$$(proc, e)$$

Synchronous Product

definition: a synchronisation constraint over the alphabets $A_1, \ldots, A_n$ of LTSs is a subset of $A_1 \times \cdots \times A_n$. Elements of synchronisation constraints are called synchronisation vectors

definition: let $\mathcal{A}_i$, $i = 1, \ldots, n$, be LTSs with alphabets A_i and let $I \subseteq A_1 \times \cdots \times A_n$ be a synchronisation constraint. The synchronous product $(\mathcal{A}_1, \ldots, \mathcal{A}_n, I)$ of the $\mathcal{A}_i$ under I is the sub-LTS of the free product $\mathcal{A}_1 \times \cdots \times \mathcal{A}_n$ that contains only those transitions $(t_1, \ldots, t_n)$ with $\lambda(t_1, \ldots, t_n) \in I$

intuition: global transitions of a synchronous product must respect the synchronisation constraints

Synchronous Product

example: synchronous product of sequential program with boolean variable

$$(1, \mathit{true}) \rightarrow_{(b=\mathit{true}?,\mathit{true}!)} (1, \mathit{true}) \qquad (2, \mathit{true}) \rightarrow_{(b:=\mathit{true},b:=\mathit{true})} (3, \mathit{true})$$

$$(3, \mathit{true}) \rightarrow_{(\mathit{proc},e)} (4, \mathit{true}) \qquad (4, \mathit{true}) \rightarrow_{(b:=\mathit{false},b:=\mathit{false})} (1, \mathit{false})$$

$$(1, \mathit{false}) \rightarrow_{(b=\mathit{false}?,\mathit{false}!)} (2, \mathit{false}) \qquad (2, \mathit{false}) \rightarrow_{(b:=\mathit{true},b:=\mathit{true})} (3, \mathit{true})$$

$$(3, \mathit{false}) \rightarrow_{(\mathit{proc},e)} (4, \mathit{false}) \qquad (4, \mathit{false}) \rightarrow_{(b:=\mathit{false},b:=\mathit{false})} (1, \mathit{false})$$

eliminating unreachable states from initial state $(1, \mathit{false})$ yields

$$(3, \mathit{true}) \rightarrow_{(\mathit{proc},e)} (4, \mathit{true}) \qquad (4, \mathit{true}) \rightarrow_{(b:=\mathit{false},b:=\mathit{false})} (1, \mathit{false})$$

$$(1, \mathit{false}) \rightarrow_{(b=\mathit{false}?,\mathit{false}!)} (2, \mathit{false}) \qquad (2, \mathit{false}) \rightarrow_{(b:=\mathit{true},b:=\mathit{true})} (3, \mathit{true})$$

Synchronous Product

example: Peterson's algorithm (cf. Arnold's book)

- the free product has 5 components: the two processes and three boolean variables for d_0, d_1 and $turn$
- the interaction of the variables in the processes cause many synchronisation constraints
- if both processes are executed on a single processor,then only one process can progress in each time interval, the other executes e
- such assumptions lead to synchronisation constraints with ~ 20 global actions
- from initial state $(1, 1, \mathit{false}, \mathit{false}, 0)$ there are ~ 75 possible transitions
- no transition has 5 as first and second component, i.e., the mutex property holds

idea: verify the mutex property by adding an observer process that in each step tests the values of the first and second component

Pragmatics of Synchronisation

example: processes interacting by shared variables

- represent each process and variable by LTS
- make constraint synchronise accesses to variables with actions of variables
- use constraints to make processes respect access policy for shared resources

Pragmatics of Synchronisation

example: two boolean variables, one observer

- observer tests for equal values
- this yields synchronisation vectors

$$(b = b'?, true!, true!)$$

$$(b = b'?, false!, false!)$$

$$(b \neq b'?, true!, false!)$$

$$(b \neq b'?, false!, true!)$$

Pragmatics of Synchronisation

example: communication by message passing

- communication channels are shared objects
- sending is synchronised with entering message into channel
- receiving is synchronised with taking a message from channel
- bounded channels can be modelled as bounded buffers (see previous example)
- lossy, duplifying or modifying buffers can be modelled similarly

remark: message passing is sufficient to model asynchronous communication between processes

Interleaving

asynchronous communication: only one process can make progress per time step

idea: consider all possible interleavings of processes

realisation:

- add appropriate e-actions to LTSs
- impose appropriate synchronisation constraints

intuition: interleaving semantics corresponds to unwinding of LTS

question: how to label edges of the tree in a systematic way?

Example: The Alternating Bit Protocol

context: the alternating bit protocol is a popular exercise in specification and verification

here: simplified version to obtain small synchronous products

specification: (informal)

- sender sends alternating bits and receives same bits as acknowledgements
- receiver receives alternating bits and sends same bits as acknowledgements
- communication is instantaneous, i.e., no message passing through buffer
- inversion of control bit modelled as error; then message or acknowledgement is resent

Example: The Alternating Bit Protocol

actions of sender:

- $m_0!$: send message labelled by 0 bit
- $m_1!$: send message labelled by 1 bit
- $a_0?$: receive acknowledgement labelled by 0 bit
- $a_1?$: receive acknowledgement labelled by 1 bit

actions of receiver:

- $m_0?$: receive message labelled by 0 bit
- $m_1?$: receive message labelled by 1 bit
- $a_0!$: send acknowledgement labelled by 0 bit
- $a_1!$: send acknowledgement labelled by 1 bit

implicit assumption: at each instant, every entity sends or receives (no e-actions)

Example: The Alternating Bit Protocol

states:

- s_0: state from which message or acknowledgement 0 is sent
- s_1: state from which message or acknowledgement 1 is sent
- r_0: state from which message or acknowledgement 0 is resent
- r_1: state from which message or acknowledgement 1 is resent
- w_0: state waiting for message or acknowledgement 0
- w_1: state waiting for message or acknowledgement 1

Example: The Alternating Bit Protocol

LTS of sender:

$$t_1 : s_1 \rightarrow_{m_1!} w_1 \qquad t_2 : s_0 \rightarrow_{m_0!} w_0 \qquad t_3 : r_1 \rightarrow_{m_1!} w_1 \qquad t_4 : r_0 \rightarrow_{m_0!} w_0$$

$$t_5 : w_0 \rightarrow_{a_0?} s_1 \qquad t_6 : w_0 \rightarrow_{a_1?} r_0 \qquad t_7 : w_1 \rightarrow_{a_1?} s_0 \qquad t_8 : w_1 \rightarrow_{a_0?} r_1$$

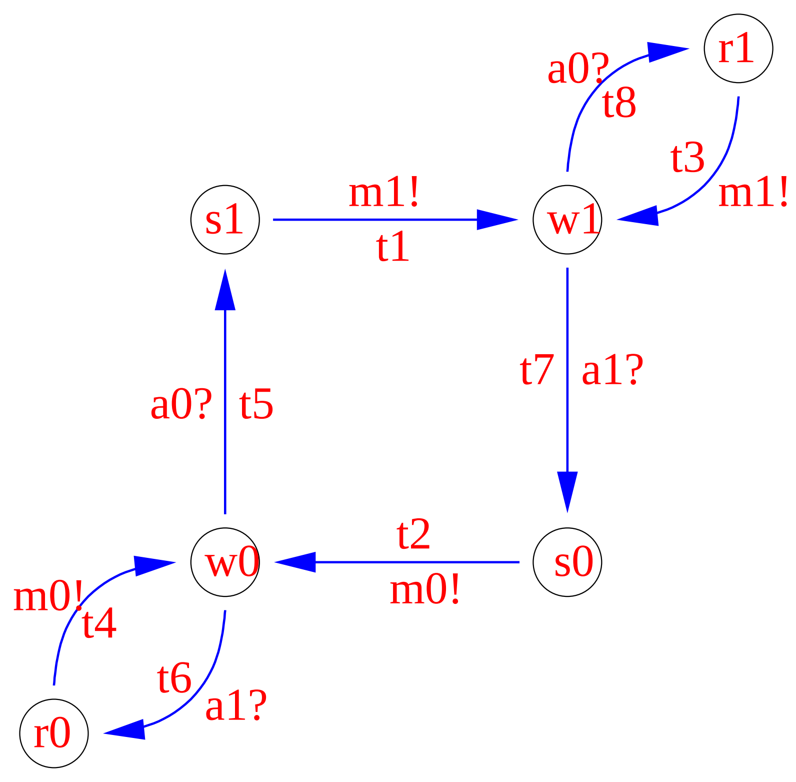
initial states: s_1, s_0

emissions: t_1, t_2

re-emission: t_3, t_4 (message was not properly acknowledged)

Example: The Alternating Bit Protocol

LTS of sender:



Example: The Alternating Bit Protocol

LTS of receiver:

$$t'_1 : w_1 \rightarrow_{m_1?} s_1 \qquad t'_2 : w_1 \rightarrow_{m_0?} r_0 \qquad t'_3 : w_0 \rightarrow_{m_0?} s_0 \qquad t'_4 : w_0 \rightarrow_{m_1?} r_1$$

$$t'_5 : s_1 \rightarrow_{a_1!} w_0 \qquad t'_6 : s_0 \rightarrow_{a_0!} w_1 \qquad t'_7 : r_1 \rightarrow_{a_1!} w_0 \qquad t'_8 : r_0 \rightarrow_{a_0!} w_1$$

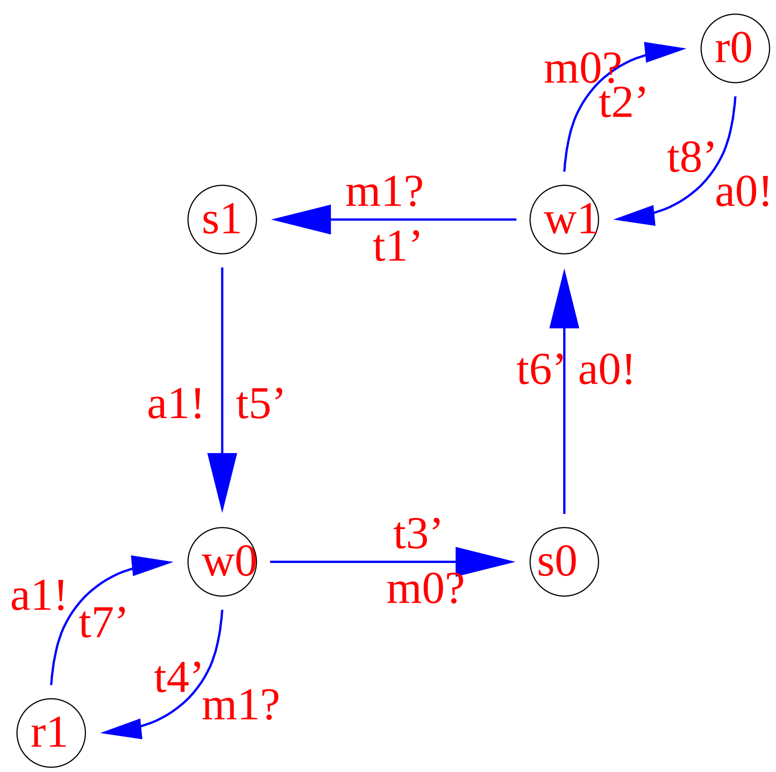
initial states: w_1, w_0

well-received: t'_1, t'_3 (control bit has expected value)

re-emission: t'_2, t'_4 (control bit has un-expected value)

Example: The Alternating Bit Protocol

LTS of receiver:



Example: The Alternating Bit Protocol

synchronisation constraints:

1. no transmission errors for messages and acknowledgements

$$\{(m_0!, m_0?), (m_1!, m_1?), (a_0?, a_0!), (a_1?, a_1!)\}$$

2. transmission errors for messages, but not for acknowledgements

$$\{(m_0!, m_0?), (m_1!, m_1?), (a_0?, a_0!), (a_1?, a_1!), (m_0!, m_1?), (m_1!, m_0?)\}$$

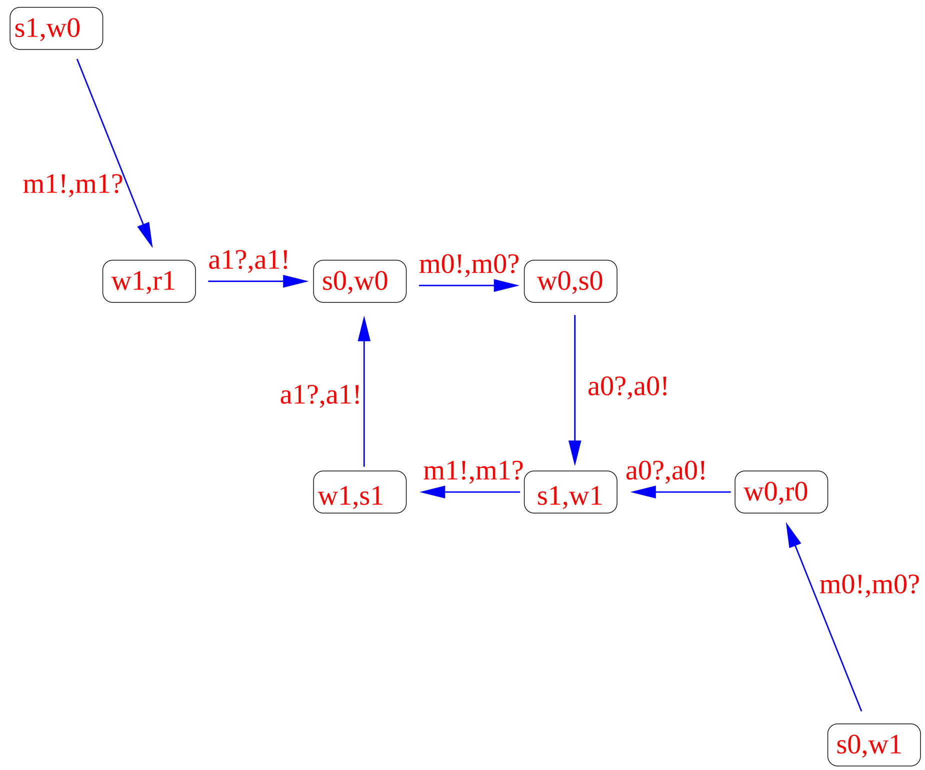
3. transmission errors for acknowledgements, but not for messages

$$\{(m_0!, m_0?), (m_1!, m_1?), (a_0?, a_0!), (a_1?, a_1!), (a_0?, a_1!), (a_1?, a_0!)\}$$

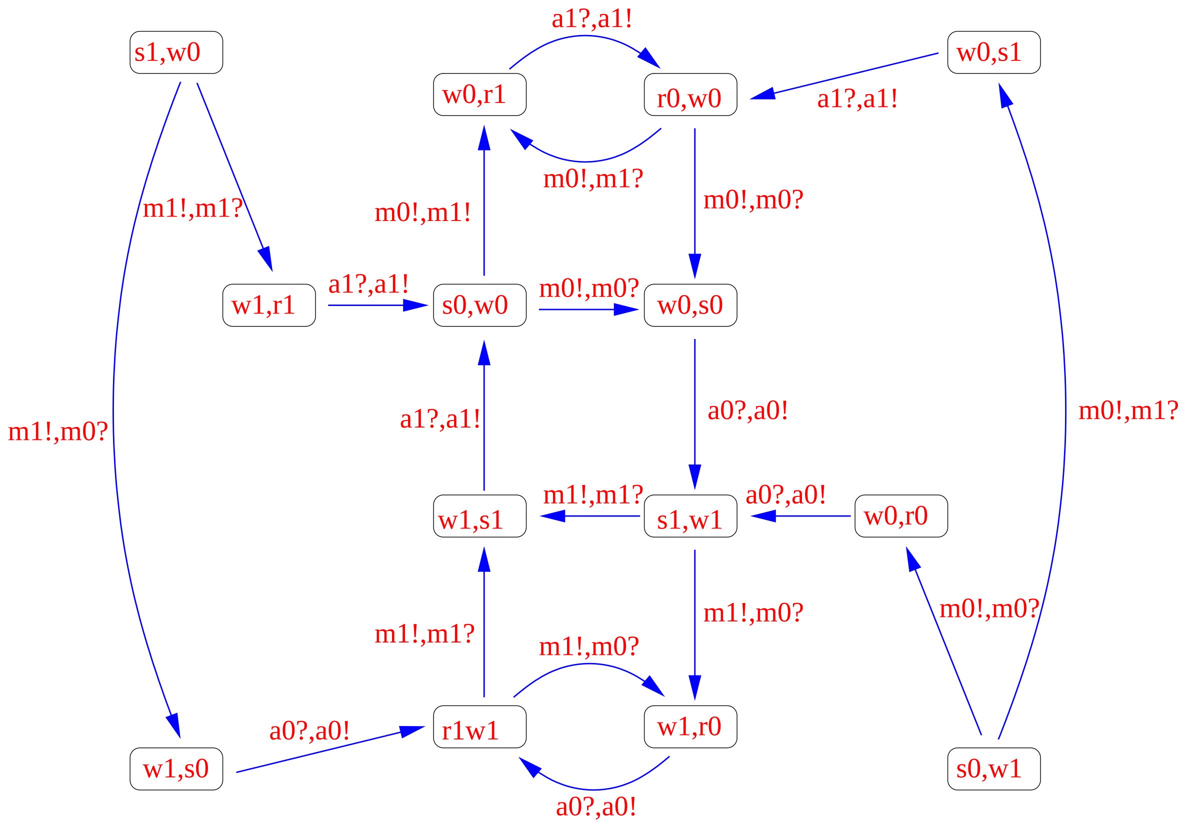
4. transmission errors for messages and acknowledgements

$$\{(m_0!, m_0?), (m_1!, m_1?), (a_0?, a_0!), (a_1?, a_1!), \\ (m_0!, m_1?), (m_1!, m_0?), (a_0?, a_1!), (a_1?, a_0!)\}$$

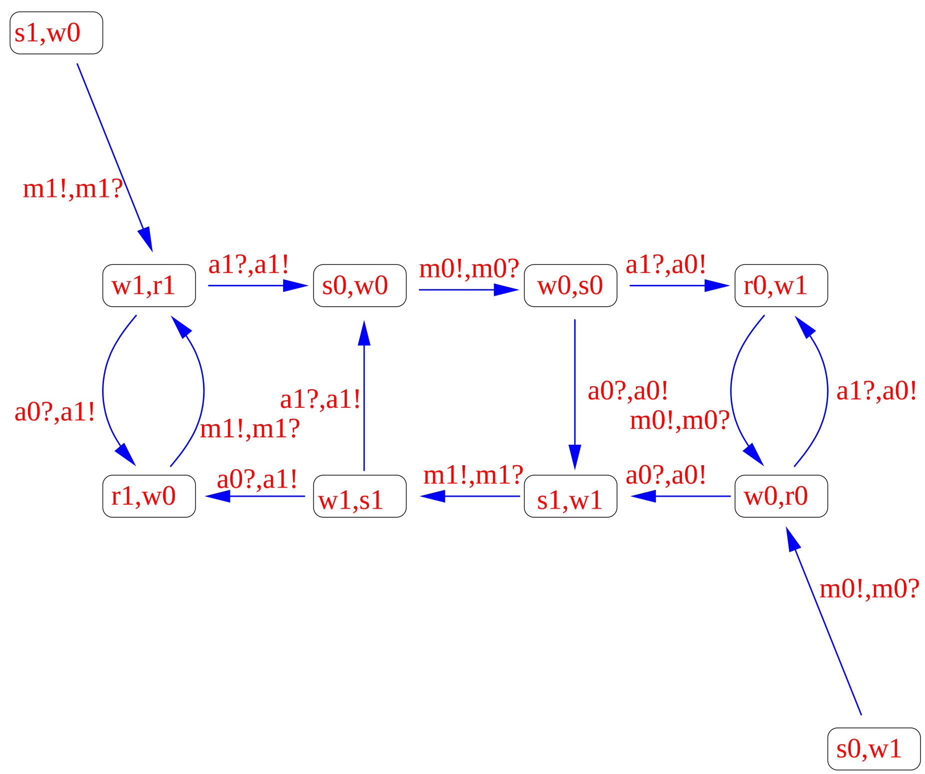
Example: The Alternating Bit Protocol



Example: The Alternating Bit Protocol



Example: The Alternating Bit Protocol



Example: The Alternating Bit Protocol