Modeling Concurrent Systems

1. A Client/Server System

- System of one server and two clients.
- Three concurrently executing system components.
- Server manages a resource.
  - An object that only one system component may use at any time.
- Clients request resource and, having received an answer, use it.
- Server ensures that not both clients use resource simultaneously.
- Server eventually answers every request.

Set of system requirements.

2. Modeling Concurrent Systems

3. A Model of the Client/Server System

4. Summary
Desired System Properties

- Property: mutual exclusion.
  - At no time, both clients are in critical region.
  - Critical region: program region after receiving resource from server and before returning resource to server.
  - The system shall only reach states, in which mutual exclusion holds.

- Property: no starvation.
  - Always when a client requests the resource, it eventually receives it.
  - Always when the system reaches a state, in which a client has requested a resource, it shall later reach a state, in which the client receives the resource.

Problem: each system component executes its own program.

Multiple program states exist at each moment in time.

Total system state is combination of individual program states.

Not easy to see which system states are possible.

How can we verify that the system has the desired properties?

1. A Client/Server System

2. Modeling Concurrent Systems

3. A Model of the Client/Server System

4. Summary

System States

At each moment in time, a system is in a particular state.

- A state \( s : \text{Var} \rightarrow \text{Val} \)
  - A state \( s \) is a mapping of every system variable \( x \) to its value \( s(x) \).
  - Typical notation: \( s = [x = 0, y = 1, \ldots] = [0, 1, \ldots] \).
  - \( \text{Var} \) ... the set of system variables
  - Program variables, program counters, ...
  - \( \text{Val} \) ... the set of variable values.

- The state space \( \text{State} = \{ s \mid s : \text{Var} \rightarrow \text{Val} \} \)
  - The state space is the set of possible states.
  - The system variables can be viewed as the coordinates of this space.
  - The state space may (or may not) be finite.
    - If \( |\text{Var}| = n \) and \( |\text{Val}| = m \), then \( |\text{State}| = mn \).
    - A word of \( \log_2 mn \) bits can represent every state.

A system execution can be described by a path \( s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow \ldots \) in the state space.

Deterministic Systems

In a sequential system, each state typically determines its successor state.

- The system is deterministic.
  - We have a (possibly not total) transition function \( F \) on states.
    - \( s_1 = F(s_0) \) means “\( s_1 \) is the successor of \( s_0 \)”.
  - Given an initial state \( s_0 \), the execution is thus determined.
    - \( s_0 \rightarrow s_1 = F(s_0) \rightarrow s_2 = F(s_1) \rightarrow \ldots \)
  - A deterministic system (model) is a pair \((I, F)\).
    - A set of initial states \( I \subseteq \text{State} \)
    - Initial state condition \( \text{I}(s) : \iff s \in I \)
    - A transition function \( F : \text{State} \rightarrow \text{State} \).
  - A run of a deterministic system \((I, F)\) is a (finite or infinite) sequence \( s_0 \rightarrow s_1 \rightarrow \ldots \) of states such that
    - \( s_0 \in I \) (respectively \( I(s_0) \)).
    - \( s_{i+1} = F(s_i) \) (for all sequence indices \( i \))
    - If \( s \) ends in a state \( s_n \), then \( F \) is not defined on \( s_n \).
Nondeterministic Systems

In a concurrent system, each component may change its local state, thus the successor state is not uniquely determined.

- The system is nondeterministic.
- We have a transition relation $R$ on states.
- $R(s_0, s_1)$ means “$s_1$ is a (possible) successor of $s_0$”.
- Given an initial state $s_0$, the execution is not uniquely determined.
- Both $s_0 \rightarrow s_1 \rightarrow \ldots$ and $s_0 \rightarrow s'_1 \rightarrow \ldots$ are possible.
- A non-deterministic system (model) is a pair $\langle I, R \rangle$.
  - A set of initial states (initial state condition) $I \subseteq \text{State}$.
  - A transition relation $R \subseteq \text{State} \times \text{State}$.
- A run $s$ of a nondeterministic system $\langle I, R \rangle$ is a (finite or infinite) sequence $s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow \ldots$ of states such that
  - $s_0 \in I$ (respectively $I(s_0)$).
  - $R(s_i, s_{i+1})$ (for all sequence indices $i$).
  - If $s$ ends in a state $s_n$, then there is no state $t$ such that $R(s_n, t)$.

Derived Notions

- Successor and predecessor:
  - State $t$ is a (direct) successor of state $s$, if $R(s, t)$.
  - State $s$ is then a predecessor of $t$.
- Reachability:
  - A state $t$ is reachable, if there exists some run $s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow \ldots$ such that $t = s_i$ (for some $i$).
  - A state $t$ is unreachable, if it is not reachable.

Reachability Graph

The transitions of a system can be visualized by a graph.

The nodes of the graph are the reachable states of the system.

Examples

Examples

- A deterministic system $W = (I_W, F_W)$ ("watch").
  - $State := \mathbb{N}_{24} \times \mathbb{N}_{60}$.
  - $I_W(h, m) := \langle h = 0 \land m = 0 \rangle$.
  - $F_W(h, m) := \langle h, m + 1 \rangle$.

- A nondeterministic system $C = (I_C, R_C)$ (modulo 3 “counter”).
  - $State := \mathbb{N}_3$.
  - $I_C(i) := i = 0$.
  - $R_C(i, i') := inc(i, i') \lor dec(i, i')$.
    - $inc(i, i') := \langle if i < 2 then i' = i + 1 else i' = 0 \rangle$.
    - $dec(i, i') := \langle if i > 0 then i' = i - 1 else i' = 2 \rangle$.

Initial States of Composed System

What are the initial states $I$ of the composed system?

- Set $I := I_0 \times \ldots \times I_{n-1}$.
  - $I_i$ is the set of initial states of component $i$.
  - Set of initial states is Cartesian product of the sets of initial states of the individual components.

- Predicate $I(s_0, \ldots, s_{n-1}) := I_0(s_0) \land \ldots \land I_{n-1}(s_{n-1})$.
  - $I_i$ is the initial state condition of component $i$.
  - Initial state condition is conjunction of the initial state conditions of the components on the corresponding projection of the state.

Size of initial state set is the product of the sizes of the initial state sets of the individual components.

Transitions of Composed System

Which transitions can the composed system perform?

- Synchronized composition.
  - At each step, every component must perform a transition.
    - $R_i$ is the transition relation of component $i$.
    - $R_i(s_i, s'_i) := \langle s = s' \land s_{\neg i} = s_{\neg i}' \rangle$.
    - $R(s_0, s_1, \ldots, s_{n-1}) := R_0(s_0, s_1) \land \ldots \land R_{n-1}(s_{n-1}, s_{n-1}')$.

- Asynchronous composition.
  - At each moment, every component may perform a transition.
    - At least one component performs a transition.
    - Multiple simultaneous transitions are possible.
    - With $n$ components, $2^n - 1$ possibilities of (combined) transitions.
      - $R(s_0, s_1, \ldots, s_{n-1}, s'_0, s'_1, \ldots, s'_{n-1}) := \langle R_0(s_0, s'_0) \land \ldots \land R_{n-1}(s_{n-1}, s'_{n-1}) \rangle$.
      - $R(s_0, s'_0) \land \ldots \land R_{n-1}(s_{n-1}, s'_{n-1})$.

Composing Systems

Compose $n$ components $S_i$ to a concurrent system $S$.

- State space $State := State_0 \times \ldots \times State_{n-1}$.
  - $State_i$ is the state space of component $i$.
  - State space is Cartesian product of component state spaces.
  - Size of state space is product of the sizes of the component spaces.

Example: three counters with state spaces $\mathbb{N}_2$ and $\mathbb{N}_3$ and $\mathbb{N}_4$. 

Fig. 1.6: The state of the product of the three counters. 

Example

System of three counters with state space $\mathbb{N}_2$ each.

- Synchronous composition:
  \[ [0, 0, 0] \iff [1, 1, 1] \]

- Asynchronous composition:

Interleaving Execution

Simplified view of asynchronous execution.

- At each moment, only one component performs a transition.
  - Do not allow simultaneous transition $t_i | t_j$ of two components $i$ and $j$.
  - Transition sequences $t_i; t_j$ and $t_j; t_i$ are possible.
  - All possible interleavings of component transitions are considered.
  - Nondeterminism is used to simulate concurrency.
  - Essentially no change of system properties.

- With $n$ components, only $n$ possibilities of a transition.

\[
R(⟨s_0, s_1, \ldots, s_{n-1}, s'_{0}, s'_{1}, \ldots, s'_{n-1}⟩) :⇔
\begin{align*}
R_0(⟨s_0, s'_{0}⟩ \land s_1 = s'_{1} \land \ldots \land s_{n-1} = s'_{n-1}) \lor \\
(s_0 = s'_0 \land R_1(⟨s_1, s'_{1}⟩ \land \ldots \land s_{n-1} = s'_{n-1}) \lor \\
\ldots \\
(s_0 = s'_0 \land s_1 = s'_1 \land \ldots \land R_{n-1}(⟨s_{n-1}, s'_{n-1}⟩).)
\end{align*}
\]

Interleaving model (respectively a variant of it) suffices in practice.

Digital Circuits

Synchronous composition of hardware components.

- A modulo 8 counter $C = ⟨l_C, R_C⟩$.

\[
\begin{align*}
\text{State} &:= \mathbb{N}_2 \times \mathbb{N}_2 \times \mathbb{N}_2. \\
l_C(v_0, v_1, v_2) :⇔ v_0 = v_1 = v_2 = 0. \\
R_C(⟨v_0, v_1, v_2⟩, ⟨v'_0, v'_1, v'_2⟩) :⇔ \\
& R_0(v_0, v'_0) \land \\
& R_1(v_0, v_1, v'_1) \land \\
& R_2(v_0, v_1, v_2, v'_2). \\
R_0(v_0, v'_0) :⇔ v'_0 = \neg v_0. \\
R_1(v_0, v_1, v'_1) :⇔ v'_1 = v_0 \oplus v_1. \\
R_2(v_0, v_1, v_2, v'_2) :⇔ v'_2 = (v_0 \land v_1) \oplus v_2.
\end{align*}
\]
Asynchronous composition of software components with shared variables.

\[ P :: l_0 : \text{while true do} \quad || \quad Q :: l_1 : \text{while true do} \]
\[ NC_0 : \text{wait turn} = 0 \quad \quad NC_1 : \text{wait turn} = 1 \]
\[ CR_0 : \text{turn} := 1 \quad \quad CR_1 : \text{turn} := 0 \]

A mutual exclusion program \( M = \langle I_M, R_M \rangle \).
\[ \text{State} := PC \times PC \times \mathbb{N}_2. \quad \text{// shared variable} \]
\[ I_M((p, q, turn)) \Rightarrow p = l_0 \land q = l_1. \]
\[ R_M((p, q, turn), (p', q', turn')) \Rightarrow \]
\[ (P((p, turn), (p', turn')) \land q' = q) \lor (Q((q, turn), (q', turn')) \land p' = p). \]

Wolfgang Schreiner
http://www.risc.jku.at
23/40

Modelling Message Passing Systems

How to model an asynchronous system without shared variables where the components communicate/synchronize by exchanging messages?

- Given a label set \( \text{Label} = \text{Int} \cup \text{Ext} \cup \text{Ext}. \)
- Disjoint sets \( \text{Int} \) and \( \text{Ext} \) of internal and external labels.
- “Anonymous” label \( \_ \) in \( \text{Int} \).
- Complementary label set \( \overline{L} := \{ l : l \in L \}. \)
- A labeled system is a pair \( \langle I, R \rangle \).
- Initial state condition \( I \subseteq \text{State}. \)
- Labeled transition relation \( R \subseteq \text{Label} \times \text{State} \times \text{State}. \)
- A run of a labeled system \( \langle I, R \rangle \) is a (finite or infinite) sequence \( s_0 \xrightarrow{I} s_1 \xrightarrow{I} \ldots \) of states such that
  - \( s_0 \in I. \)
  - \( R(I, s_i, s_{i+1}) \) (for all sequence indices \( i \)).
  - If \( s \) ends in a state \( s_n \), there is no label \( I \) and state \( t \) s.t. \( R(I, s_n, t) \).
Synchronization by Message Passing

Compose a set of $n$ labeled systems $\langle I_i, R_i \rangle$ to a system $\langle I, R \rangle$.

- **State space** $State := State_0 \times \ldots \times State_{n-1}$.
- **Initial states** $I := I_0 \times \ldots \times I_{n-1}$.
  - $I(s_0, \ldots, s_{n-1}) \iff I_0(s_0) \land \ldots \land I_{n-1}(s_{n-1})$.
- **Transition relation**
  
  $R(I, \langle s_i \rangle_{i \in \mathbb{N}}, \langle s'_i \rangle_{i \in \mathbb{N}}) \iff$
  
  $(\exists i \in \mathbb{N} : R_i(I, s_i, s'_i) \land \forall k \in \mathbb{N} \setminus \{i\} : s_k = s'_k) \lor$
  
  $(I = l \land \exists i \in \text{Ext}, i \in \mathbb{N}, j \in \mathbb{N} : R_i(I, s_i, s'_i) \land R_i(I, s_j, s'_j) \land \forall k \in \mathbb{N} \setminus \{i, j\} : s_k = s'_k)$.

Either a component performs an internal transition or two components simultaneously perform an external transition with complementary labels.

Example (Continued)

Composition of $\langle I_0, R_0 \rangle$ and $\langle I_1, R_1 \rangle$ to $\langle I, R \rangle$.

State $= (PC \times \mathbb{N}) \times (PC \times \mathbb{N})$.

$I(p, i, q, j) :\iff p = a_0 \land i \in \mathbb{N} \land q = b_0$.

$R(I, (p, i, q, j), (p', i', q', j')) :\iff$

$(I = A \land (p = a_2 \land p' = a_0 \land i' = i + 1) \land (q' = q \land j' = j)) \lor$

$(I = B \land (p' = p \land i' = i) \land (q = b_1 \land q' = b_2 \land j' = j + 1)) \lor$

$(I = A \land (p = a_0 \land p' = a_1 \land i' = i) \land (q = b_0 \land q' = b_1 \land j' = j)) \lor$

$(I = (p, i, q, j)) :\iff$

Problem: state relation of each component refers to local variable of other component (variables are shared).

Communication by Message Passing

Example (Revised)

Two labeled systems $\langle I_0, R_0 \rangle$ and $\langle I_1, R_1 \rangle$.

- **External** $:= \{M_k : k \in \mathbb{N}\} \cup \{N_k : k \in \mathbb{N}\}$.
- $R_0(I, (p, i, j), (p', i', j')) :\iff$

  $(I = M \land p = a_0 \land p' = a_1 \land i' = i) \lor$

  $(\exists k \in \mathbb{N} : I = N_k \land p = a_1 \land p' = a_0 \land i' = k) \lor$

  $(I = A \land p = a_2 \land p' = a_0 \land i' = i + 1)$. 

- $R_1(I, (q, j), (q', j')) :\iff$

  $(\exists k \in \mathbb{N} : I = M_k \land q = b_0 \land q' = b_1 \land j' = k) \lor$

  $(I = B \land q = b_2 \land q' = b_2 \land j' = j + 1) \lor$

  $(I = N \land q = b_2 \land q' = b_0 \land j' = j)$.

Encode message value in label.
Example (Continued)

Composition of \( \langle I_0, R_0 \rangle \) and \( \langle I_1, R_1 \rangle \) to \( \langle I, R \rangle \).

\[
\text{State} = (PC \times \mathbb{N}) \times (PC \times \mathbb{N}).
\]

\[
I(p, i, q, j) :\iff p = a_0 \land i \in \mathbb{N} \land q = b_0.
\]

\[
R(I, (p, i, q, j), (p', i', q', j')) :\iff
\begin{align*}
& (I = A \land (p = a_2 \land p' = a_0 \land i' = i + 1) \land (q' = q \land j' = j)) \lor \\
& (I = B \land (p' = p \land i' = i) \land (q = b_1 \land q' = b_2 \land j' = j + 1)) \lor \\
& (I = \_ \land \_ \land k \in \mathbb{N} : k = i \land \\
& (p = a_0 \land p' = a_1 \land i' = i) \land (q = b_0 \land q' = b_1 \land j' = k)) \lor \\
& (I = \_ \land \_ \land k \in \mathbb{N} : k = j \land \\
& (p = a_1 \land p' = a_2 \land i' = k) \land (q = b_2 \land q' = b_0 \land j' = j)).
\end{align*}
\]

Logically equivalent to previous definition of transition relation.

The Client/Server System

Asynchronous composition of three components \( \text{Client}_1, \text{Client}_2, \text{Server} \).

- **Client**: \( \text{State} := PC \times \mathbb{N}_2 \times \mathbb{N}_2 \).
  - Three variables \( pc, \text{request}, \text{answer} \).
  - \( pc \) represents the program counter.
  - \( \text{request} \) is the buffer for outgoing requests.
    - Filled by client, when a request is to be sent to server.
  - \( \text{answer} \) is the buffer for incoming answers.
    - Checked by client, when it waits for an answer from the server.
- **Server**: \( \text{State} := (\mathbb{N}_3)^3 \times (\{1, 2\} \rightarrow \mathbb{N}_2)^2 \).
  - Variables \( \text{given}, \text{waiting}, \text{sender}, rbuffer, sbuffer \).
  - No program counter.
    - We use the value of \( \text{sender} \) to check whether server waits for a request (\( \text{sender} = 0 \)) or answers a request (\( \text{sender} \neq 0 \)).
  - Variables \( \text{given}, \text{waiting}, \text{sender} \) as in program.
  - \( rbuffer(i) \) is the buffer for incoming requests from client \( i \).
  - \( sbuffer(i) \) is the buffer for outgoing answers to client \( i \).

External Transitions

\[
\text{Ext} := \{ \text{REQ}_1, \text{REQ}_2, \text{ANS}_1, \text{ANS}_2 \}.
\]

- Transition labeled \( \text{REQ}_i \) transmits a request from client \( i \) to server.
  - Enabled when \( \text{request} \neq 0 \) in client \( i \).
  - Effect in client \( i \) : \text{request}' = 0.
  - Effect in server: \( rbuffer'(i) = 1 \).
- Transition labeled \( \text{ANS}_i \) transmits an answer from server to client \( i \)
  - Enabled when \( sbuffer(i) \neq 0 \).
  - Effect in server: \( sbuffer'(i) = 0 \).
  - Effect in client \( i \) : \text{answer}' = 1.

The external transitions correspond to system-level actions of the communication subsystem (rather than to the user-level actions of the client/server program).
The Client

Client system $C_i = \langle IC_i, RC_i \rangle$.

State := $(PC \times N_3 \times N_2)$.

Int := $(S_i, S, C_i)$.

$IC_i(pc, request, answer) \iff$
$pc = R \land request = 0 \land answer = 0$.

$RC_i(l, \langle pc, request, answer \rangle),$
$\langle pc', request', answer' \rangle) :=$
$l = W \land pc = 0 \land request = 0 \land$
$pc' = pc \land request' = 1 \land answer' = answer \lor$
$l = A_1 \land pc = 0 \land request = 0 \land$
$pc' = pc \land request' = request \land answer' = 0 \lor$
$l = A_2 \land pc = 0 \land request = 0 \land$
$pc' = pc \land request' = request \land answer' = 0 \lor$
$l = A_3 \land pc = 0 \land request = 0 \land$
$pc' = pc \land request' = request \land answer' = 1 \lor$
$l = F \land pc' = 0 \land request' = 0 \land answer' = 0 \land$
$pc = \emptyset \land W := pc \land waiting := 0 \land$
$given := 0; waiting := 0$.

Client(ident): param ident begin loop ...
$R: sendRequest()$
$S: receiveAnswer()$
$C: / critical region$
... sendRequest() endloop end Client

The Server

Server system $S = \langle IS, RS \rangle$.

State := $(N_4^3 \times \{1, 2\} \to N_2^2)$.

Int := $(\{D, D_2, F, A_1, A_2, W\})$.

$IS(given, waiting, sender, rbuffer, sbuffer) \iff$
$given = waiting = sender = 0 \land$
$rbuffer(1) = rbuffer(2) = sbuffer(1) = sbuffer(2) = 0$.

$RS(l, \langle given, waiting, sender, rbuffer, sbuffer \rangle,)$
$\langle given', waiting', sender', rbuffer', sbuffer' \rangle) :=$
$\exists i \in \{1, 2\} :$
$l = D \land given = 0 \land rbuffer(i) = 0 \land$
$sender = i \land sbuffer(i) = 0 \land$
$U(given, waiting, sbuffer) \land$
$\forall j \in \{1, 2\} \setminus \{i\} : U_j(rbuffer) \lor$
... $U_i(x_1, \ldots, x_n) := x_1 = x_1 \land \ldots \land x_n = x_n$.

Server:
local given, waiting, sender
begin
$given := 0; waiting := 0$
loop
$D: given := receiveRequest()$
if given = then
$waiting' := sender \land waiting := 0 \land$
$U(given, rbuffer, sbuffer) \lor$
...$i \in \{1, 2\} :$
$l = W \land given = 0 \land rbuffer(i) = 0 \land$
$sender = i \land sbuffer(i) = 0 \land$
$U(given, waiting, rbuffer) \land$
$\forall j \in \{1, 2\} \setminus \{i\} : U_j(sbuffer)$.

$D: given := receiveRequest()$
if given = then
$waiting := 0$
else
$A_1: given := waiting; waiting := 0$
sendAnswer(given)
endif
elsi given = then
$A_2: given := sender$
sendAnswer(given)
else
$W: waiting := sender$
sendAnswer(given)
endif
endloop end Server
Communication Channels

We also model the communication medium between components.

- **Bounded channel** \( \text{Channel}_{i,j} = (\text{ICH}, RCH_{i,j}) \).
  - Transfers message from component with address \( i \) to component \( j \).
  - May hold at most \( N \) messages at a time (for some \( N \)).

**State**: \( \text{State} := \text{Value}^* \).
  - Sequence of values of type \( \text{Value} \).

**Ext**: \( \{ \text{SEND}_{i,j}(m) : m \in \text{Value} \} \cup \{ \text{RECEIVE}_{i,j}(m) : m \in \text{Value} \} \).

By \( \text{SEND}_{i,j}(m) \), channel receives from sender \( i \) a message \( m \) destined for receiver \( j \); by \( \text{RECEIVE}_{i,j}(m) \), channel forwards that message.

**ICH(queue)** \( \iff \exists m \in \text{Value} : (l = \text{SEND}_{i,j}(m) \land |\text{queue}| < N \land \text{queue}' = \text{queue} \circ (m)) \lor (l = \text{RECEIVE}_{i,j}(m) \land |\text{queue}| > 0 \land \text{queue} = (m) \circ \text{queue}'). \)

Client/Server Example with Channels

- Server receives address 0.
  - Label \( \text{REQ}_i \) is renamed to \( \text{RECEIVE}_{i,0}(R) \).
  - Label \( \text{ANS}_i \) is renamed to \( \text{SEND}_{0,i}(A) \).
- Client \( i \) receives address \( i \) (\( i \in \{1, 2\} \)).
  - Label \( \text{REQ}_i \) is renamed to \( \text{SEND}_{i,0}(R) \).
  - Label \( \text{ANS}_i \) is renamed to \( \text{RECEIVE}_{0,i}(A) \).
- System is composed of seven components:
  - \( \text{Server, Client}_1, \text{Client}_2 \).
  - \( \text{Channel}_{0,1}, \text{Channel}_{1,0} \).
  - \( \text{Channel}_{0,2}, \text{Channel}_{2,0} \).

Also channels are active system components.

Summary

- A system is described by
  - its (finite or infinite) state space,
  - the initial state condition (set of input states),
  - the transition relation on states.
- State space of composed system is product of component spaces.
  - Variable shared among components occurs only once in product.
- System composition can be
  - synchronous: conjunction of individual transition relations.
    - Suitable for digital hardware.
  - asynchronous: disjunction of relations.
    - Interleaving model: each relation conjoins the transition relation of one component with the identity relations of all other components.
    - Suitable for concurrent software.
- Message passing systems may be modeled by using labels:
  - Synchronize transitions of sender and receiver.
  - Carry values to be transmitted from sender to receiver.