Modeling Concurrent Systems

Wolfgang Schreiner
Wolfgang.Schreiner@risc.jku.at

Research Institute for Symbolic Computation (RISC)
Johannes Kepler University, Linz, Austria
http://www.risc.jku.at
1. A Client/Server System

2. Modeling Concurrent Systems

3. A Model of the Client/Server System

4. Summary
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.
System Implementation

Server:
local given, waiting, sender
begin
given := 0; waiting := 0
loop
  sender := receiveRequest()
  if sender = given then
    if waiting = 0 then
      given := 0
      sendAnswer(given)
    else
      given := waiting; waiting := 0
      sendAnswer(given)
    endif
  elseif given = 0 then
    given := sender
    sendAnswer(given)
  else
    waiting := sender
  endif
endloop
end Server

Client(ident):
param ident
begin
loop
  ...
  sendRequest()
  receiveAnswer()
  ... // critical region
  sendRequest()
endloop
end Client
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 \( |\text{State}| = m^n \).
    - A word of \( \log_2 m^n \) 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 $\langle I, F \rangle$.
  - A set of initial states $I \subseteq \text{State}$
    - Initial state condition $I(s) :\iff s \in I$
  - A transition function $F : \text{State}^{\text{partial}} \rightarrow \text{State}$.
- A run of a deterministic system $\langle I, F \rangle$ 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 \( s_0 \rightarrow s_1 \rightarrow s_2 \ldots \) 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 $t$.
  - A finite run $s_0 \rightarrow \ldots \rightarrow s_n$ ends in a state which has no successor.

- **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.

Not all states are reachable (typically most are unreachable).
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

1. Automata

Fig. 1.1. A model of a watch

of $A_{c3}$ correspond to the possible counter values. Its transitions reflect the possible actions on the counter. In this example we restrict our operations to increments (inc) and decrements (dec).

Fig. 1.2. $A_{c3}$: a modulo 3 counter

Examples

- A deterministic system $W = (I_W, F_W)$ ("watch").
  - $State := \mathbb{N}_{24} \times \mathbb{N}_{60}$.
  - $\mathbb{N}_n := \{ i \in \mathbb{N} : i < n \}$.
  - $I_W(h, m) : \iff h = 0 \land m = 0$.
  - $I_W := \{ \langle h, m \rangle : h = 0 \land m = 0 \} = \{ \langle 0, 0 \rangle \}$.
  - $F_W(h, m) :=$
    \begin{align*}
    &\text{if } m < 59 \text{ then } \langle h, m + 1 \rangle \\
    &\text{else if } h < 23 \text{ then } \langle h + 1, 0 \rangle \\
    &\text{else } \langle 0, 0 \rangle.
    \end{align*}

- A nondeterministic system $C = (I_C, R_C)$ (modulo 3 "counter").
  - $State := \mathbb{N}_3$.
  - $I_C(i) : \iff i = 0$.
  - $R_C(i, i') : \iff inc(i, i') \lor dec(i, i')$.
    - $inc(i, i') : \iff$ if $i < 2$ then $i' = i + 1$ else $i' = 0$.
    - $dec(i, i') : \iff$ if $i > 0$ then $i' = i - 1$ else $i' = 2$. 
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 \( \mathbb{N}_4 \).

![Diagram of the states of the product of the three counters](image)

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** $I(s_0, \ldots, s_{n-1}) :\iff I_0(s_0) \land \ldots \land I_{n-1}(s_{n-1})$.
  - $I_i$ 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 \( i \).
  
  \[
  R\left(\langle s_0, \ldots, s_{n-1}\rangle, \langle s_0', \ldots, s_{n-1}'\rangle\right) : \iff \\
  R_0(s_0, s_0') \land \ldots \land R_{n-1}(s_{n-1}, s_{n-1}').
  \]

- **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.

\[
R\left(\langle s_0, \ldots, s_{n-1}\rangle, \langle s_0', \ldots, s_{n-1}'\rangle\right) : \iff \\
(R_0(s_0, s_0') \land \ldots \land s_{n-1} = s_{n-1}') \lor \\
\ldots \\
(s_0 = s_0' \land \ldots \land R_{n-1}(s_{n-1}, s_{n-1}')) \lor \\
\ldots \\
(R_0(s_0, s_0') \land \ldots \land R_{n-1}(s_{n-1}, s_{n-1}')).
\]
Example

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

- **Synchronous composition:**
  \[ [0, 0, 0] \leftrightarrow [1, 1, 1] \]

- **Asynchronous composition:**

Fig. 1.10. A few transitions of the product of the three counters

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.

$$R(⟨s_0, s_1, \ldots, s_{n-1}⟩, ⟨s'_0, s'_1, \ldots, s'_{n-1}⟩) \overset{∀}{=} (R_0(s_0, s'_0) \land s_1 = s'_1 \land \ldots \land s_{n-1} = s'_{n-1}) \lor$$
$$((s_0 = s'_0 \land R_1(s_1, s'_1) \land \ldots \land s_{n-1} = s'_{n-1}) \lor$$
$$\ldots$$
$$((s_0 = s'_0 \land s_1 = s'_1 \land \ldots \land R_{n-1}(s_{n-1}, s'_{n-1})).$$

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

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

Synchronous composition of hardware components.

- A modulo 8 counter $C = \langle I_C, R_C \rangle$.

  State := $\mathbb{N}_2 \times \mathbb{N}_2 \times \mathbb{N}_2$.

  $I_C(v_0, v_1, v_2) \iff v_0 = v_1 = v_2 = 0$.

  $R_C(\langle v_0, v_1, v_2 \rangle, \langle v'_0, v'_1, v'_2 \rangle) \iff$
  \begin{align*}
  & 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).
  \end{align*}

  $R_0(v_0, v'_0) \iff v'_0 = \neg v_0$.
  $R_1(v_0, v_1, v'_1) \iff v'_1 = v_0 \oplus v_1$.
  $R_2(v_0, v_1, v_2, v'_2) \iff v'_2 = (v_0 \land v_1) \oplus v_2$.

Concurrent Software

Asynchronous composition of software components with shared variables.

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

\[
\text{A mutual exclusion program} \ M = \langle I_M, R_M \rangle.
\]

\[
\text{State} := PC \times PC \times \mathbb{N}_2. \ // \text{shared variable}
\]
\[
l_M(p, q, turn) :\Leftrightarrow p = l_0 \land q = l_1.
\]
\[
R_M(\langle p, q, turn \rangle, \langle p', q', turn' \rangle) :\Leftrightarrow
\]
\[
(\text{P(\langle p, turn \rangle, \langle p', turn' \rangle) } \land q' = q) \lor (\text{Q(\langle q, turn \rangle, \langle q', turn' \rangle) } \land p' = p).
\]

\[
\begin{align*}
P(\langle p, turn \rangle, \langle p', turn' \rangle) & :\Leftrightarrow \\
& (p = l_0 \land p' = NC_0 \land turn' = turn) \lor \\
& (p = NC_0 \land p' = CR_0 \land turn = 0 \land turn' = turn) \lor \\
& (p = CR_0 \land p' = l_0 \land turn' = 1).
\end{align*}
\]

\[
\begin{align*}
Q(\langle q, turn \rangle, \langle q', turn' \rangle) & :\Leftrightarrow \\
& (q = l_1 \land q' = NC_1 \land turn' = turn) \lor \\
& (q = NC_1 \land q' = CR_1 \land turn = 1 \land turn' = turn) \lor \\
& (q = CR_1 \land q' = l_1 \land turn' = 0).
\end{align*}
\]
Concurrent Software

Figure 2.2
Reachable states of Kripke structure for mutual exclusion example.


Model guarantees mutual exclusion.
Modeling Commands

Transition relations are typically described in a particular form.

- \( R(s, s') \iff P(s) \land s' = F(s) \).
  - Guard condition \( P \) on state in which transition can be performed.
    - If \( P(s) \) holds, then there exists some \( s' \) such that \( R(s, s') \).
  - Transition function \( F \) that determines the successor of \( s \).
    - \( F \) is defined for all states for which \( P(s) \) holds:
      \( F : \{ s \in \text{State} : P(s) \} \rightarrow \text{State} \).

- Examples:
  - Assignment: \( l : x := e; \ m : \ldots \)
    - \( R(\langle pc, x, y \rangle, \langle pc', x', y' \rangle) :\iff pc = l \land (x' = e \land y' = y \land pc' = m) \).
  - Wait statement: \( l : \textbf{wait} \ P(x, y); \ m : \ldots \)
    - \( R(\langle pc, x, y \rangle, \langle pc', x', y' \rangle) :\iff pc = l \land P(x, y) \land (x' = x \land y' = y \land pc' = m) \).
  - Guarded assignment: \( l : P(x, y) \rightarrow x := e; \ m : \ldots \)
    - \( R(\langle pc, x, y \rangle, \langle pc', x', y' \rangle) :\iff pc = l \land P(x, y) \land (x' = e \land y' = y \land pc' = m) \).

Most programming language commands can be translated into this form.
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 $\text{Label} = \text{Int} \cup \text{Ext} \cup \overline{\text{Ext}}$.
  - Disjoint sets Int and Ext of internal and external labels.
  - “Anonymous” label $\perp \in \text{Int}$.
  - Complementary label set $\overline{L} := \{\overline{l} : l \in L\}$.

- 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 $s_0 \xrightarrow{l_0} s_1 \xrightarrow{l_1} \ldots$ of states such that
  - $s_0 \in I$.
  - $R(l_i, s_i, s_{i+1})$ (for all sequence indices $i$).
  - If $s$ ends in a state $s_n$, there is no label $l$ and state $t$ s.t. $R(l, s_n, t)$. 

Wolfgang Schreiner
http://www.risc.jku.at
Synchronization by Message Passing

Compose a set of \( n \) labeled systems \( \langle l_i, R_i \rangle \) to a system \( \langle l, R \rangle \).

- **State space** \( \text{State} := \text{State}_0 \times \ldots \times \text{State}_{n-1} \).
- **Initial states** \( I := l_0 \times \ldots \times l_{n-1} \).
  - \( I(s_0, \ldots, s_{n-1}) :\Leftrightarrow l_0(s_0) \wedge \ldots \wedge l_{n-1}(s_{n-1}) \).
- **Transition relation**
  \[
  R(l, \langle s_i \rangle_{i \in \mathbb{N}_n}, \langle s'_i \rangle_{i \in \mathbb{N}_n}) \Leftrightarrow \\
  (l \in \text{Int} \wedge \exists i \in \mathbb{N}_n : R_i(l, s_i, s'_i) \wedge \forall k \in \mathbb{N}_n \setminus \{i\} : s_k = s'_k) \vee \\
  (l = _{-} \wedge \exists l \in \text{Ext}, i \in \mathbb{N}_n, j \in \mathbb{N}_n : \\
  R_i(l, s_i, s'_i) \wedge R_j(l, s_j, s'_j) \wedge \forall k \in \mathbb{N}_n \setminus \{i, j\} : s_k = s'_k). 
  \]

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

0 :: loop
    a₀ : send(i)
    a₁ : i := receive()
    a₂ : i := i + 1
end

1 :: loop
    b₀ : j := receive()
    b₁ : j := j + 1
    b₂ : send(j)
end

Two labeled systems \( \langle I₀, R₀ \rangle \) and \( \langle I₁, R₁ \rangle \).

\( State₀ = State₁ = PC \times \mathbb{N} \), Internal := \( \{A, B\} \), External := \( \{M, N\} \).

\( I₀(p,i) \iff p = a₀ \land i \in \mathbb{N}; \ I₁(q,j) \iff q = b₀.\)

\( R₀(l, \langle p,i \rangle, \langle p',i' \rangle) \iff\)
\( (l = M \land p = a₀ \land p' = a₁ \land i' = i) \lor\)
\( (l = N \land p = a₁ \land p' = a₂ \land i' = j) \lor \quad \text{// illegal!}\)
\( (l = A \land p = a₂ \land p' = a₀ \land i' = i + 1).\)

\( R₁(l, \langle q,j \rangle, \langle q',j' \rangle) \iff\)
\( (l = M \land q = b₀ \land q' = b₁ \land j' = i) \lor \quad \text{// illegal!}\)
\( (l = B \land q = b₁ \land q' = b₂ \land j' = j + 1) \lor\)
\( (l = N \land q = b₂ \land q' = b₀ \land j' = j).\)
Example (Continued)

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

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

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

\[
R(l, \langle p, i, q, j \rangle, \langle p', i', q', j' \rangle) \iff
\begin{align*}
(l = A \land (p = a_2 \land p' = a_0 \land i' = i + 1) \land (q' = q \land j' = j)) & \lor \\
(l = B \land (p' = p \land i' = i) \land (q = b_1 \land q' = b_2 \land j' = j + 1)) & \lor \\
(l = - \land (p = a_0 \land p' = a_1 \land i' = i) \land (q = b_0 \land q' = b_1 \land j' = i)) & \lor \\
(l = - \land (p = a_1 \land p' = a_2 \land i' = j) \land (q = b_2 \land q' = b_0 \land j' = j)).
\end{align*}
\]

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

Two labeled systems $\langle l_0, R_0 \rangle$ and $\langle l_1, R_1 \rangle$.

\[ R_0(l, \langle p, i \rangle, \langle p', i' \rangle) : \iff \]
\[ (l = \overline{M_i} \land p = a_0 \land p' = a_1 \land i' = i) \lor \]
\[ (\exists k \in \mathbb{N} : l = N_k \land p = a_1 \land p' = a_2 \land i' = k) \lor \]
\[ (l = A \land p = a_2 \land p' = a_0 \land i' = i + 1). \]

\[ R_1(l, \langle q, j \rangle, \langle q', j' \rangle) : \iff \]
\[ (\exists k \in \mathbb{N} : l = M_k \land q = b_0 \land q' = b_1 \land j' = k) \lor \]
\[ (l = B \land q = b_1 \land q' = b_2 \land j' = j + 1) \lor \]
\[ (l = \overline{N_j} \land q = b_2 \land q' = b_0 \land j' = j). \]

Encode message value in label.
Example (Continued)

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

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

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

$R(l, \langle p, i, q, j \rangle, \langle p', i', q', j' \rangle) :\iff$

- $(l = A \land (p = a_2 \land p' = a_0 \land i' = i + 1) \land (q' = q \land j' = j)) \lor$
- $(l = B \land (p' = p \land i' = i) \land (q = b_1 \land q' = b_2 \land j' = j + 1)) \lor$
- $(l = \_ \land \exists k \in \mathbb{N} : k = i \land$
  - $(p = a_0 \land p' = a_1 \land i' = i) \land (q = b_0 \land q' = b_1 \land j' = k)) \lor$
- $(l = \_ \land \exists k \in \mathbb{N} : k = j \land$
  - $(p = a_1 \land p' = a_2 \land i' = k) \land (q = b_2 \land q' = b_0 \land j' = j))$.

Logically equivalent to previous definition of transition relation.
1. A Client/Server System

2. Modeling Concurrent Systems

3. A Model of the Client/Server System

4. Summary
The Client/Server System

Asynchronous composition of three components \(Client_1, Client_2, Server\).

- **Client\(_i\):** \(State := PC \times \mathbb{N}_2 \times \mathbb{N}_2\).
  - Three variables \(pc, request, answer\).
  - \(pc\) represents the program counter.
  - \(request\) is the buffer for outgoing requests.
    - Filled by client, when a request is to be sent to server.
  - \(answer\) is the buffer for incoming answers.
    - Checked by client, when it waits for an answer from the server.

- **Server:** \(State := (\mathbb{N}_3)^3 \times (\{1, 2\} \rightarrow \mathbb{N}_2)^2\).
  - Variables \(given, waiting, sender, rbuffer, sbuffer\).
  - No program counter.
    - We use the value of \(sender\) to check whether server waits for a request \((sender = 0)\) or answers a request \((sender \neq 0)\).
  - Variables \(given, waiting, sender\) as in program.
  - \(rbuffer(i)\) is the buffer for incoming requests from client \(i\).
  - \(sbuffer(i)\) is the buffer for outgoing answers to client \(i\).

Wolfgang Schreiner http://www.risc.jku.at
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: \( \text{rbuffer}'(i) = 1 \).
  - Transition labeled \( \text{ANS}_i \) transmits an answer from server to client \( i \)
    - Enabled when \( \text{sbuffer}(i) \neq 0 \).
    - Effect in server: \( \text{sbuffer}'(i) = 0 \).
    - 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 :\( = PC \times \mathbb{N}_2 \times \mathbb{N}_2 \).

Int :\( = \{ R_i, S_i, C_i \} \).

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

\( RC_i(l, \langle pc, request, answer \rangle, \langle pc', request', answer' \rangle) :\iff \)
\( \quad (l = R_i \land pc = R \land request = 0 \land \)
\( \quad \quad pc' = S \land request' = 1 \land answer' = answer) \lor \)
\( \quad (l = S_i \land pc = S \land answer \neq 0 \land \)
\( \quad \quad pc' = C \land request' = request \land answer' = 0) \lor \)
\( \quad (l = C_i \land pc = C \land request = 0 \land \)
\( \quad \quad pc' = R \land request' = 1 \land answer' = answer) \lor \)
\( \quad (l = REQ_i \land request \neq 0 \land \)
\( \quad \quad pc' = pc \land request' = 0 \land answer' = answer) \lor \)
\( \quad (l = ANS_i \land \)
\( \quad \quad pc' = pc \land request' = request \land answer' = 1). \)

Client(ident):
\( \quad \text{param ident} \)
begin
\( \quad \text{loop} \)
\( \quad \quad \ldots \)
\( \quad R: \text{sendRequest()} \)
\( \quad S: \text{receiveAnswer()} \)
\( \quad C: // \text{critical region} \)
\( \quad \quad \ldots \)
\( \quad \quad \text{sendRequest()} \)
endloop
end Client
The Server

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

State := $(\mathbb{N}_3)^3 \times (\{1, 2\} \to \mathbb{N}_2)^2$.

Int := \{D1, D2, F, A1, A2, W\}.

IS\((\text{given}, \text{waiting}, \text{sender}, rbuffer, sbuffer)\) \iff
\begin{align*}
given &= \text{waiting} = \text{sender} = 0 \land \\
rbuffer(1) &= rbuffer(2) = sbuffer(1) = sbuffer(2) = 0.
\end{align*}

RS\((l, \langle \text{given}, \text{waiting}, \text{sender}, rbuffer, sbuffer \rangle), \\
\langle \text{given}', \text{waiting}', \text{sender}', rbuffer', sbuffer' \rangle\) \iff
\exists i \in \{1, 2\} : 
\begin{align*}
l &= D_i \land \text{sender} = 0 \land rbuffer(i) \neq 0 \land \\
\text{sender}' &= i \land rbuffer'(i) = 0 \land \\
U(\text{given}, \text{waiting}, sbuffer) \land \\
\forall j \in \{1, 2\} \setminus \{i\} : U_j(rbuffer)) \lor \\
\cdots
\end{align*}

\begin{align*}
U(x_1, \ldots, x_n) &\iff x'_1 = x_1 \land \ldots \land x'_n = x_n, \\
U_j(x_1, \ldots, x_n) &\iff x'_1(j) = x_1(j) \land \ldots \land x'_n(j) = x_n(j).
\end{align*}

Server:
local given, waiting, sender
begin
given := 0; waiting := 0
loop
D: sender := receiveRequest()
if sender = given then
if waiting = 0 then
F: given := 0
else
A1: given := waiting;
waiting := 0
sendAnswer(given)
endif
elseif given = 0 then
A2: given := sender
sendAnswer(given)
else
W: waiting := sender
endif
endloop
end Server
The Server (Contd)

\[ l = F \land sender \neq 0 \land sender = given \land waiting = 0 \land given' = 0 \land sender' = 0 \land U(waiting, rbuffer, sbuffer) \lor \]

\[ l = A1 \land sender \neq 0 \land sbuffer(waiting) = 0 \land 
    sender = given \land waiting \neq 0 \land 
    given' = waiting \land waiting' = 0 \land 
    sbuffer'(waiting) = 1 \land sender' = 0 \land 
    U(rbuffer) \land 
    \forall j \in \{1, 2\}\{waiting\} : U_j(sbuffer) \lor \]

\[ l = A2 \land sender \neq 0 \land sbuffer(sender) = 0 \land 
    sender \neq given \land given = 0 \land 
    given' = sender \land 
    sbuffer'(sender) = 1 \land sender' = 0 \land 
    U(waiting, rbuffer) \land 
    \forall j \in \{1, 2\}\{sender\} : U_j(sbuffer) \lor \]

Server:
local given, waiting, sender
begin
    given := 0; waiting := 0
loop
    D: sender := receiveRequest()
        if sender = given then
            if waiting = 0 then
                F: given := 0
                endif
                else
                    A1: given := waiting;
                        waiting := 0
                        sendAnswer(given)
                    endif
                    elsif given = 0 then
                        A2: given := sender
                            sendAnswer(given)
                    else
                        W: waiting := sender
                            endif
                    endloop
end Server
The Server (Contd’2)

\[\ldots\]
\[(l = W \land sender \neq 0 \land sender \neq given \land given \neq 0 \land waiting' := sender \land sender' = 0 \land U(given, rbuffer, sbuffer)) \lor\]

\[
\exists i \in \{1, 2\}:
\]
\[
(l = REQ_i \land rbuffer'(i) = 1 \land U(given, waiting, sender, sbuffer) \land \forall j \in \{1, 2\} \setminus \{i\} : U_j(rbuffer)) \lor
\]
\[
(l = ANS_i \land sbuffer(i) \neq 0 \land sbuffer'(i) = 0 \land U(given, waiting, sender, rbuffer) \land \forall j \in \{1, 2\} \setminus \{i\} : U_j(sbuffer)).
\]

Server:

local given, waiting, sender
begin
  given := 0; waiting := 0
loop
  D: sender := receiveRequest()
  if sender = given then
    if waiting = 0 then
      F: given := 0
    else
      A1: given := waiting;
      waiting := 0
      sendAnswer(given)
    endif
  endif
  if given = 0 then
    A2: given := sender
    sendAnswer(given)
  else
    W: waiting := sender
  endif
endloop
end Server
Communication Channels

We also model the communication medium between components.

- **Bounded channel** \( \text{Channel}_{i,j} = (ICH, RCH_{i,j}) \).
  - Transfers message from component with address \( i \) to component \( j \).
  - May hold at most \( N \) messages at a time (for some \( N \)).
- **State** \( := \text{Value}^* \).
  - Sequence of values of type \( \text{Value} \).
- **Ext** \( := \{SEND_{i,j}(m) : m \in \text{Value}\} \cup \{RECEIVE_{i,j}(m) : m \in \text{Value}\} \).
  - By \( SEND_{i,j}(m) \), channel receives from sender \( i \) a message \( m \) destined for receiver \( j \); by \( RECEIVE_{i,j}(m) \), channel forwards that message.

\[
ICH(\text{queue}) :\iff \text{queue} = \langle \rangle.
\]
\[
RCH_{i,j}(l, \text{queue}, \text{queue}') :\iff \\
\exists m \in \text{Value} : \\
(l = SEND_{i,j}(m) \land |\text{queue}| < N \land \text{queue}' = \text{queue} \circ \langle m \rangle) \lor \\
(l = \overline{RECEIVE}_{i,j}(m) \land |\text{queue}| > 0 \land \text{queue} = \langle m \rangle \circ \text{queue}') .
\]
Client/Server Example with Channels

- Server receives address 0.
  - Label $REQ_i$ is renamed to $RECEIVE_{i,0}(R)$.
  - Label $ANS_i$ is renamed to $SEND_{0,i}(A)$.

- Client $i$ receives address $i$ ($i \in \{1, 2\}$).
  - Label $REQ_i$ is renamed to $SEND_{i,0}(R)$.
  - Label $ANS_i$ is renamed to $RECEIVE_{0,i}(A)$.

- System is composed of seven components:
  - Server, $Client_1$, $Client_2$.
  - $Channel_{0,1}$, $Channel_{1,0}$.
  - $Channel_{0,2}$, $Channel_{2,0}$.

Also channels are active system components.
1. A Client/Server System

2. Modeling Concurrent Systems

3. A Model of the Client/Server System

4. 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.