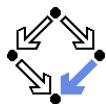


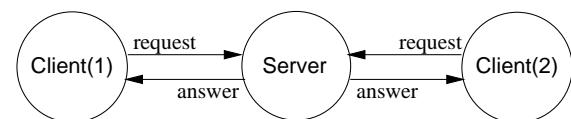
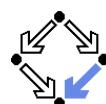
Modeling Concurrent Systems

Wolfgang Schreiner
Wolfgang.Schreiner@risc.uni-linz.ac.at

Research Institute for Symbolic Computation (RISC)
Johannes Kepler University, Linz, Austria
<http://www.risc.uni-linz.ac.at>



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.

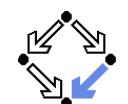
Set of system requirements.

1. A Client/Server System

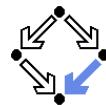
2. Modeling Concurrent Systems

3. A Model of the Client/Server System

System Implementation



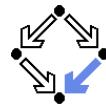
```
Server:  
  local given, waiting, sender  
begin  
  given := 0; waiting := 0  
  loop  
    sender := receiveRequest()  
    if sender = given then  
      if waiting = 0 then  
        given := 0  
      else  
        given := waiting; waiting := 0  
        sendAnswer(given)  
      endif  
    elsif 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?

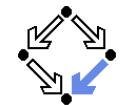


System States

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: $s = [x = 0, y = 1, \dots] = [0, 1, \dots]$.
 - Var ... the set of system variables
 - Program variables, program counters, ...
 - 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.

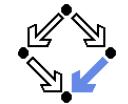
A system execution can be described by a path $s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow \dots$ in the state space.



1. A Client/Server System

2. Modeling Concurrent Systems

3. A Model of the Client/Server System

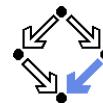


Deterministic Systems

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.
 - $s_0 \rightarrow s_1 = F(s_0) \rightarrow s_2 = F(s_1) \rightarrow \dots$
- A **deterministic system (model)** is a pair $\langle I, F \rangle$.
 - A set of initial states $I \subseteq \text{State}$
 - **Initial state condition** $I(s) \Leftrightarrow s \in I$
 - A transition function $F : \text{State} \xrightarrow{\text{partial}} \text{State}$.
- A **run** of a deterministic system $\langle I, F \rangle$ is a (finite or infinite) sequence $s_0 \rightarrow s_1 \rightarrow \dots$ 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 .

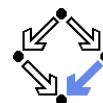
Nondeterministic Systems



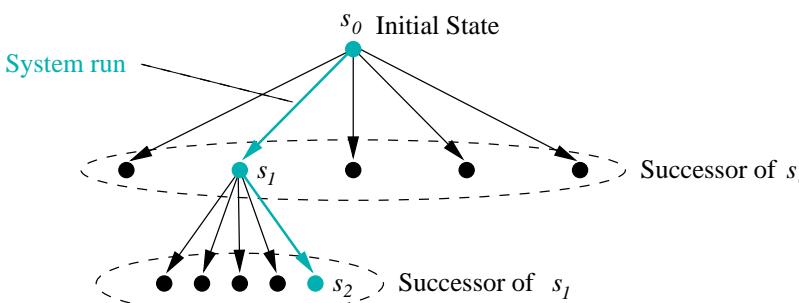
In a concurrent system, each component may change its local state, thus the successor state is not uniquely determined.

- 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 $s_0 \rightarrow s_1 \rightarrow \dots$ and $s_0 \rightarrow s'_1 \rightarrow \dots$ 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 $s_0 \rightarrow s_1 \rightarrow s_2 \dots$ 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 $R(s_n, t)$.

Reachability Graph

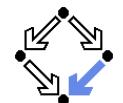


The transitions of a system can be visualized by a graph.



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

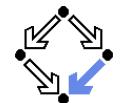
Derived Notions



- 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 $s_0 \rightarrow \dots \rightarrow s_n$ ends in a state which has no successor.
- Reachability:
 - A state t is **reachable**, if there exists some run $s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow \dots$ 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).

Examples



6 1. Automata

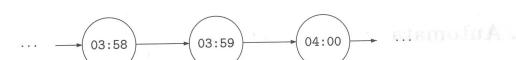


Fig. 1.1. A model of a watch

of \mathcal{A}_{ct} 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).

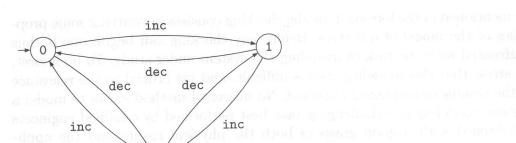
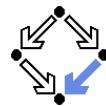


Fig. 1.2. \mathcal{A}_{ct} : a modulo 3 counter using a state transition diagram. The states are numbered 0, 1, 2. The transitions are labeled inc (increment) and dec (decrement). The automaton starts at state 0 and cycles through states 0, 1, 2, 0, 1, 2, ...



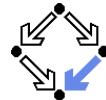
Examples

- A deterministic system $W = (I_W, F_W)$ ("watch").

- $State := \{(h, m) : h \in \mathbb{N}_{24} \wedge m \in \mathbb{N}_{60}\}$.
 - $\mathbb{N}_n := \{i \in \mathbb{N} : i < n\}$.
 - $I_W(h, m) :\Leftrightarrow h = 0 \wedge m = 0$.
 - $I_W := \{(h, m) : h = 0 \wedge m = 0\} = \{(0, 0)\}$.
 - $F_W(h, m) :=$
 - $\text{if } m < 59 \text{ then } \langle h, m + 1 \rangle$
 - $\text{else if } h < 24 \text{ then } \langle h + 1, 0 \rangle$
 - $\text{else } \langle 0, 0 \rangle$.

- A nondeterministic system $C = (I_C, R_C)$ (modulo 3 "counter").

- $State := \mathbb{N}_3$.
 - $I_C(i) :\Leftrightarrow i = 0$.
 - $R_C(i, i') :\Leftrightarrow inc(i, i') \vee dec(i, i')$.
 - $inc(i, i') :\Leftrightarrow \text{if } i < 2 \text{ then } i' = i + 1 \text{ else } i' = 0$.
 - $dec(i, i') :\Leftrightarrow \text{if } i > 0 \text{ then } i' = i - 1 \text{ else } i' = 2$.

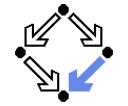


Initial States of Composed System

What are the initial states I of the composed system?

- Set** $I := I_0 \times \dots \times I_{n-1}$.
- 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** $I(s_0, \dots, s_{n-1}) :\Leftrightarrow I_0(s_0) \wedge \dots \wedge I_{n-1}(s_{n-1})$.
- 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.



Composing Systems

Compose n components S_i to a concurrent system S .

- State space** $State := State_0 \times \dots \times State_{n-1}$.
- $State_i$ 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 .

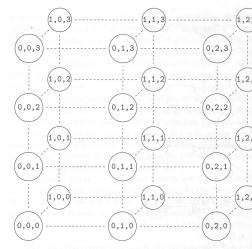
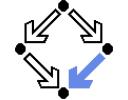


Fig. 1.9. The states of the product of the three counters

B.Berard et al: "Systems and Software Verification", 2001.

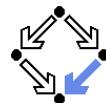


Transitions of Composed System

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 .
$$R(\langle s_0, \dots, s_{n-1} \rangle, \langle s'_0, \dots, s'_{n-1} \rangle) :\Leftrightarrow R_0(s_0, s'_0) \wedge \dots \wedge R_{n-1}(s_{n-1}, s'_{n-1}).$$
- 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.
$$R(\langle s_0, \dots, s_{n-1} \rangle, \langle s'_0, \dots, s'_{n-1} \rangle) :\Leftrightarrow (R_0(s_0, s'_0) \wedge \dots \wedge s_{n-1} = s'_{n-1}) \vee \\ \dots \\ (s_0 = s'_0 \wedge \dots \wedge R_{n-1}(s_{n-1}, s'_{n-1})) \vee \\ \dots \\ (R_0(s_0, s'_0) \wedge \dots \wedge R_{n-1}(s_{n-1}, s'_{n-1})).$$

Example



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

- Synchronous composition:

$$[0, 0, 0] \xrightarrow{\quad} [1, 1, 1]$$

- Asynchronous composition:

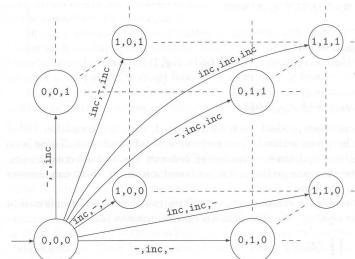


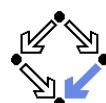
Fig. 1.10. A few transitions of the product of the three counters

B.Berard et al: "Systems and Software Verification", 2001.

Wolfgang Schreiner

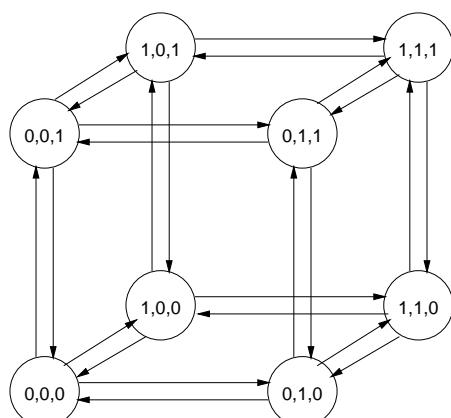
<http://www.risc.uni-linz.ac.at>

17/39



Example

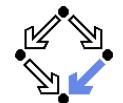
System of three counters with state space \mathbb{N}_2 each.



Wolfgang Schreiner

<http://www.risc.uni-linz.ac.at>

19/39



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.

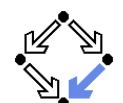
$$\begin{aligned} R(\langle s_0, s_1, \dots, s_{n-1} \rangle, \langle s'_0, s'_1, \dots, s'_{n-1} \rangle) :&\Leftrightarrow \\ &(R_0(s_0, s'_0) \wedge s_1 = s'_1 \wedge \dots \wedge s_{n-1} = s'_{n-1}) \vee \\ &(s_0 = s'_0 \wedge R_1(s_1, s'_1) \wedge \dots \wedge s_{n-1} = s'_{n-1}) \vee \\ &\dots \\ &(s_0 = s'_0 \wedge s_1 = s'_1 \wedge \dots \wedge R_{n-1}(s_{n-1}, s'_{n-1})). \end{aligned}$$

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

Wolfgang Schreiner

<http://www.risc.uni-linz.ac.at>

18/39



Digital Circuits

Synchronous composition of hardware components.

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

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

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

$$\begin{aligned} R_C(\langle v_0, v_1, v_2 \rangle, \langle v'_0, v'_1, v'_2 \rangle) :&\Leftrightarrow \\ &R_0(v_0, v'_0) \wedge \\ &R_1(v_0, v_1, v'_1) \wedge \\ &R_2(v_0, v_1, v_2, v'_2). \end{aligned}$$

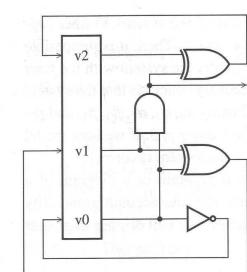


Figure 2.1
Synchronous modulo 8 counter.

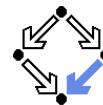
Edmund Clarke et al: "Model Checking", 1999.

Wolfgang Schreiner

<http://www.risc.uni-linz.ac.at>

20/39

Concurrent Software



Asynchronous composition of software components with shared variables.

```
P :: l0 : while true do          ||   Q :: l1 : while true do
    NC0 : wait turn = 0           NC1 : wait turn = 1
    CR0 : turn := 1               CR1 : turn := 0
  end                                end
```

- A mutual exclusion program $M = \langle I_M, R_M \rangle$.

State := PC × PC × N₂. // shared variable

$I_M(p, q, turn) :\Leftrightarrow p = l_0 \wedge q = l_1$.

$R_M(\langle p, q, turn \rangle, \langle p', q', turn' \rangle) :\Leftrightarrow$

$(P(\langle p, turn \rangle, \langle p', turn' \rangle) \wedge q' = q) \vee (Q(\langle q, turn \rangle, \langle q', turn' \rangle) \wedge p' = p)$.

$P(\langle p, turn \rangle, \langle p', turn' \rangle) :\Leftrightarrow$

$(p = l_0 \wedge p' = NC_0 \wedge turn' = turn) \vee$

$(p = NC_0 \wedge p' = CR_0 \wedge turn = 0 \wedge turn' = turn) \vee$

$(p = CR_0 \wedge p' = l_0 \wedge turn' = 1)$.

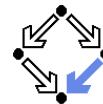
$Q(\langle q, turn \rangle, \langle q', turn' \rangle) :\Leftrightarrow$

$(q = l_1 \wedge q' = NC_1 \wedge turn' = turn) \vee$

$(q = NC_1 \wedge q' = CR_1 \wedge turn = 1 \wedge turn' = turn) \vee$

$(q = CR_1 \wedge q' = l_1 \wedge turn' = 0)$.

Modeling Commands



Transition relations are typically described in a particular form.

- $R(s, s') :\Leftrightarrow P(s) \wedge s' = F(s)$.
 - Precondition 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 s holds:

$$F : \{s \in State : P(s)\} \rightarrow State$$
- Examples:
 - Assignment: $I : x := e; m : \dots$
 - $R(\langle x, y \rangle, \langle x', y' \rangle) :\Leftrightarrow pc = I \wedge (x' = e \wedge y' = y \wedge pc' = m)$.
 - Wait statement: $I : \text{wait } P(x, y); m : \dots$
 - $R(\langle x, y \rangle, \langle x', y' \rangle) :\Leftrightarrow pc = I \wedge P(x, y) \wedge (x' = x \wedge y' = y \wedge pc' = m)$.
 - Guarded assignment: $I : P(x, y) \rightarrow x := e; m : \dots$
 - $R(\langle x, y \rangle, \langle x', y' \rangle) :\Leftrightarrow pc = I \wedge P(x, y) \wedge (x' = e \wedge y' = y \wedge pc' = m)$.

Most programming language commands can be translated into this form.

Concurrent Software

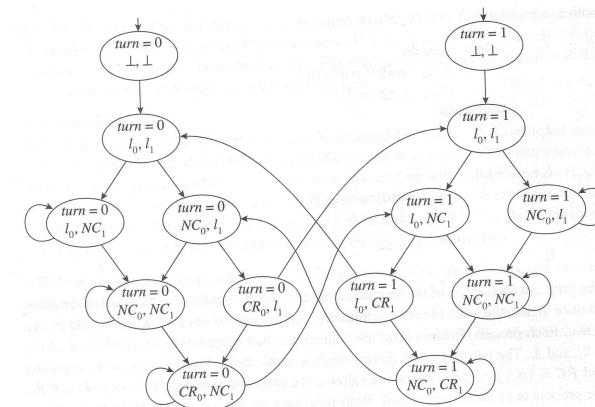
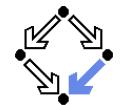
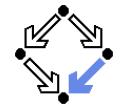


Figure 2.2
Reachable states of Kripke structure for mutual exclusion example.

Edmund Clarke et al: "Model Checking", 1999.

Model guarantees mutual exclusion.

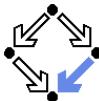
Message Passing Systems



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

- Given a label set $\text{Label} = \text{Int} \cup \text{Ext} \cup \bar{\text{Ext}}$.
 - Disjoint sets Int and Ext of internal and external labels.
 - "Anonymous" label $_I \in \text{Int}$.
 - Complementary label set $\bar{L} := \{\bar{l} : l \in L\}$.
- 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 $s_0 \xrightarrow{l_0} s_1 \xrightarrow{l_1} \dots$ 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. $R(l, s_n, t)$.

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** $State := State_0 \times \dots \times State_{n-1}$.

■ **Initial states** $I := I_0 \times \dots \times I_{n-1}$.

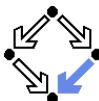
■ $I(s_0, \dots, s_{n-1}) \Leftrightarrow I_0(s_0) \wedge \dots \wedge I_{n-1}(s_{n-1})$.

■ **Transition relation**

$$R(I, \langle s_i \rangle_{i \in \mathbb{N}_n}, \langle s'_i \rangle_{i \in \mathbb{N}_n}) \Leftrightarrow \\ (I \in Int \wedge \exists i \in \mathbb{N}_n : \\ R_i(I, s_i, s'_i) \wedge \forall k \in \mathbb{N}_n \setminus \{i\} : s_k = s'_k) \vee \\ (I = - \wedge \exists I' \in Ext, i \in \mathbb{N}_n, j \in \mathbb{N}_n : \\ R_i(I, s_i, s'_i) \wedge R_j(\bar{I}, s_j, s'_j) \wedge \forall k \in \mathbb{N}_n \setminus \{i, j\} : s_k = s'_k).$$

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

Example (Continued)



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

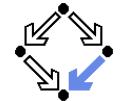
$State = (PC \times \mathbb{N}) \times (PC \times \mathbb{N})$.

$I(p, i, q, j) \Leftrightarrow p = a_0 \wedge i \in \mathbb{N} \wedge q = b_0$.

$$R(I, \langle p, i, q, j \rangle, \langle p', i', q', j' \rangle) \Leftrightarrow \\ (I = A \wedge (p = a_2 \wedge p' = a_0 \wedge i' = i + 1) \wedge (q' = q \wedge j' = j)) \vee \\ (I = B \wedge (p' = p \wedge i' = i) \wedge (q = b_1 \wedge q' = b_2 \wedge j' = j + 1)) \vee \\ (I = - \wedge (p = a_0 \wedge p' = a_1 \wedge i' = i) \wedge (q = b_0 \wedge q' = b_1 \wedge j' = i)) \vee \\ (I = - \wedge (p = a_1 \wedge p' = a_2 \wedge i' = j) \wedge (q = b_2 \wedge q' = b_0 \wedge j' = j)).$$

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

Example



0 :: **loop**

$a_0 : \text{send}(i)$
 $a_1 : i := \text{receive}()$
 $a_2 : i := i + 1$

1 :: **loop**

$b_0 : j := \text{receive}()$
 $b_1 : j := j + 1$
 $b_2 : \text{send}(j)$

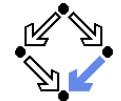
end

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

$State_0 = State_1 = PC \times \mathbb{N}$, $Internal := \{A, B\}$, $External := \{M, N\}$.
 $I_0(p, i) \Leftrightarrow p = a_0 \wedge i \in \mathbb{N}$; $I_1(q, j) \Leftrightarrow q = b_0$.

$$R_0(I, \langle p, i \rangle, \langle p', i' \rangle) \Leftrightarrow \\ (I = \bar{M} \wedge p = a_0 \wedge p' = a_1 \wedge i' = i) \vee \\ (I = N \wedge p = a_1 \wedge p' = a_2 \wedge i' = j) \vee // \text{illegal!} \\ (I = A \wedge p = a_2 \wedge p' = a_0 \wedge i' = i + 1). \\ R_1(I, \langle q, j \rangle, \langle q', j' \rangle) \Leftrightarrow \\ (I = M \wedge q = b_0 \wedge q' = b_1 \wedge j' = i) \vee // \text{illegal!} \\ (I = B \wedge q = b_1 \wedge q' = b_2 \wedge j' = j + 1) \vee \\ (I = \bar{N} \wedge q = b_2 \wedge q' = b_0 \wedge j' = j).$$

Example (Revised)



0 :: **loop**

$a_0 : \text{send}(i)$
 $a_1 : i := \text{receive}()$
 $a_2 : i := i + 1$

1 :: **loop**

$b_0 : j := \text{receive}()$
 $b_1 : j := j + 1$
 $b_2 : \text{send}(j)$

end

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

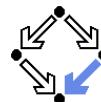
...
 $External := \{M_k : k \in \mathbb{N}\} \cup \{N_k : k \in \mathbb{N}\}$.
 $R_0(I, \langle p, i \rangle, \langle p', i' \rangle) \Leftrightarrow$

$$(I = \bar{M}_i \wedge p = a_0 \wedge p' = a_1 \wedge i' = i) \vee \\ (\exists k \in \mathbb{N} : I = N_k \wedge p = a_1 \wedge p' = a_2 \wedge i' = k) \vee \\ (I = A \wedge p = a_2 \wedge p' = a_0 \wedge i' = i + 1).$$

$$R_1(I, \langle q, j \rangle, \langle q', j' \rangle) \Leftrightarrow \\ (\exists k \in \mathbb{N} : I = M_k \wedge q = b_0 \wedge q' = b_1 \wedge j' = k) \vee \\ (I = B \wedge q = b_1 \wedge q' = b_2 \wedge j' = j + 1) \vee \\ (I = \bar{N}_j \wedge q = b_2 \wedge q' = b_0 \wedge j' = j).$$

Encode message value in label.

Example (Continued)



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

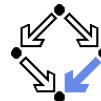
$$State = (PC \times \mathbb{N}) \times (PC \times \mathbb{N}).$$

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

$$\begin{aligned} R(I, \langle p, i, q, j \rangle, \langle p', i', q', j' \rangle) :\Leftrightarrow \\ (I = A \wedge (p = a_2 \wedge p' = a_0 \wedge i' = i + 1) \wedge (q' = q \wedge j' = j)) \vee \\ (I = B \wedge (p' = p \wedge i' = i) \wedge (q = b_1 \wedge q' = b_2 \wedge j' = j + 1)) \vee \\ (I = _ \wedge \exists k \in \mathbb{N} : k = i \wedge \\ (p = a_0 \wedge p' = a_1 \wedge i' = i) \wedge (q = b_0 \wedge q' = b_1 \wedge j' = k)) \vee \\ (I = _ \wedge \exists k \in \mathbb{N} : k = j \wedge \\ (p = a_1 \wedge p' = a_2 \wedge i' = k) \wedge (q = b_2 \wedge q' = b_0 \wedge j' = j)). \end{aligned}$$

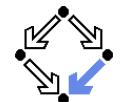
Logically equivalent to previous definition of transition relation.

Basic Idea



Asynchronous composition of three components $Client_1$, $Client_2$, $Server$.

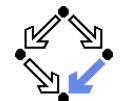
- $Client_i$: $State := PC \times \mathbb{N}_2 \times \mathbb{N}_2$.
 - Three variables pc , $request$, $answer$.
 - pc represents the program counter.
 - $request$ is the buffer for outgoing requests.
 - Filled by client, when a request is to be sent to server.
 - $answer$ is the buffer for incoming 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 .



1. A Client/Server System

2. Modeling Concurrent Systems

3. A Model of the Client/Server System



External Transitions

- $Ext := \{REQ_1, REQ_2, ANS_1, ANS_2\}$.
 - Transition labeled REQ_i transmits a request from client i to server.
 - Enabled when $request \neq 0$ in client i .
 - Effect in client i : $request' = 0$.
 - Effect in server: $rbuffer'(i) = 1$.
 - Transition labeled 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 : $answer' = 1$.

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

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

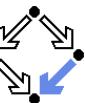
$Int := \{R_i, S_i, C_i\}$.

$IC_i(pc, request, answer) \Leftrightarrow$
 $pc = R \wedge request = 0 \wedge answer = 0.$

$RC_i(I, \langle pc, request, answer \rangle,$
 $\langle pc', request', answer' \rangle) \Leftrightarrow$
 $(I = R_i \wedge pc = R \wedge request = 0 \wedge$
 $pc' = S \wedge request' = 1 \wedge answer' = answer) \vee$
 $(I = S_i \wedge pc = S \wedge answer \neq 0 \wedge$
 $pc' = C \wedge request' = request \wedge answer' = 0) \vee$
 $(I = C_i \wedge pc = C \wedge request = 0 \wedge$
 $pc' = R \wedge request' = 1 \wedge answer' = answer) \vee$
 $(I = \overline{REQ}_i \wedge request \neq 0 \wedge$
 $pc' = pc \wedge request' = 0 \wedge answer' = answer) \vee$
 $(I = \overline{ANS}_i \wedge$
 $pc' = pc \wedge request' = request \wedge answer' = 1).$

Wolfgang Schreiner

<http://www.risc.uni-linz.ac.at>



The Server

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

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

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

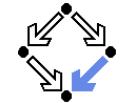
$IS(given, waiting, sender, rbuffer, sbuffer) \Leftrightarrow$
 $given = waiting = sender = 0 \wedge$
 $rbuffer(1) = rbuffer(2) = sbuffer(1) = sbuffer(2) = 0.$

$RS(I, \langle given, waiting, sender, rbuffer, sbuffer \rangle,$
 $\langle given', waiting', sender', rbuffer', sbuffer' \rangle) \Leftrightarrow$
 $\exists i \in \{1,2\} :$
 $(I = D_i \wedge sender = 0 \wedge rbuffer(i) \neq 0 \wedge$
 $sender' = i \wedge rbuffer'(i) = 0 \wedge$
 $U(given, waiting, sbuffer) \wedge$
 $\forall j \in \{1,2\} \setminus \{i\} : U_j(rbuffer)) \vee$
 \dots

$U(x_1, \dots, x_n) \Leftrightarrow x'_1 = x_1 \wedge \dots \wedge x'_n = x_n.$
 $U_j(x_1, \dots, x_n) \Leftrightarrow x'_1(j) = x_1(j) \wedge \dots \wedge x'_n(j) = x_n(j).$

Wolfgang Schreiner

<http://www.risc.uni-linz.ac.at>



The Server (Contd)

\dots

$(I = F \wedge sender \neq 0 \wedge sender = given \wedge waiting = 0 \wedge$
 $given' = 0 \wedge sender' = 0 \wedge$
 $U(waiting, rbuffer, sbuffer)) \vee$

$(I = A1 \wedge sender \neq 0 \wedge sbuffer(waiting) = 0 \wedge$
 $sender = given \wedge waiting \neq 0 \wedge$
 $given' = waiting \wedge waiting' = 0 \wedge$
 $sbuffer'(waiting) = 1 \wedge sender' = 0 \wedge$
 $U(rbuffer) \wedge$
 $\forall j \in \{1,2\} \setminus \{waiting\} : U_j(sbuffer)) \vee$

$(I = A2 \wedge sender \neq 0 \wedge sbuffer(sender) = 0 \wedge$
 $sender \neq given \wedge given = 0 \wedge$
 $given' = sender \wedge$
 $sbuffer'(sender) = 1 \wedge sender' = 0 \wedge$
 $U(waiting, rbuffer) \wedge$
 $\forall j \in \{1,2\} \setminus \{sender\} : U_j(sbuffer)) \vee$

\dots

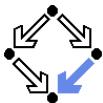
Server:

```

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

```

33/39



The Server (Contd'2)

\dots

$(I = W \wedge sender \neq 0 \wedge sender \neq given \wedge given \neq 0 \wedge$
 $waiting' := sender \wedge sender' = 0 \wedge$
 $U(given, rbuffer, sbuffer)) \vee$

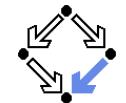
$\exists i \in \{1,2\} :$

$(I = \overline{REQ}_i \wedge rbuffer'(i) = 1 \wedge$
 $U(given, waiting, sender, sbuffer) \wedge$
 $\forall j \in \{1,2\} \setminus \{i\} : U_j(rbuffer)) \vee$

$(I = \overline{ANS}_i \wedge sbuffer(i) \neq 0 \wedge$
 $sbuffer'(i) = 0 \wedge$
 $U(given, waiting, sender, rbuffer) \wedge$
 $\forall j \in \{1,2\} \setminus \{i\} : U_j(sbuffer)).$

Wolfgang Schreiner

<http://www.risc.uni-linz.ac.at>



Server:

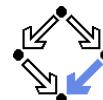
```

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

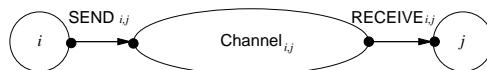
```

34/39

Communication Channels



We also model the communication medium between components.



Bounded channel $Channel_{i,j} = (ICH, RCH)$.

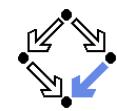
- Transfers message from component with address *i* to component *j*.
 - May hold at most *N* messages at a time (for some *N*).
- State := $\langle Value \rangle$.
 - Sequence of values of type *Value*.
- Ext := $\{SEND_{i,j}(m) : m \in Value\} \cup \{RECEIVE_{i,j}(m) : m \in Value\}$.
 - By $SEND_{i,j}(m)$,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

$ICH(queue) : \leftrightarrow queue = \langle \rangle$.

$RCH(I, queue, queue') : \leftrightarrow$

$\exists i \in Address, j \in Address, m \in Value :$

$$(I = SEND_{i,j}(m) \wedge |queue| < N \wedge queue' = queue \circ \langle m \rangle) \vee \\ (I = RECEIVE_{i,j}(m) \wedge |queue| > 0 \wedge queue = \langle m \rangle \circ queue').$$



Client/Server Example with Channels

- Server receives address 0.

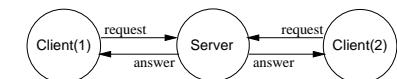
- Label REQ_i is renamed to $RECEIVE_{i,0}(R)$.
- Label ANS_i is renamed to $SEND_{0,i}(A)$.

- 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, Client₁, Client₂.
- Channel_{0,1}, Channel_{1,0}.
- Channel_{0,2}, Channel_{2,0}.



Also channels are active system components.



Summary

We have now seen a model of a client/server system.

- 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.
- **Labels** may be introduced for synchronization/communication.
 - Simultaneous transition of two components.
 - Label may describe value to be communicated.