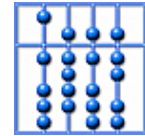


---

## **Vorlesung Specification of Distributed Systems**

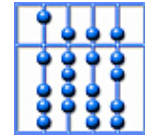
Dr. Bernhard Schätz  
Leopold-Franzens Universität Innsbruck  
Sommersemester 2005



## Overview

---

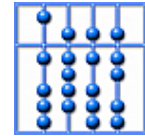
1. Introduction
2. Basics: Behavior, Interaction, Concurrency
3. Coroutines
4. Data Flow Models
5. Communicating Processes
6. State-Based Models
7. Coordination
8. Executions
9. Property Descriptions



## Overview

---

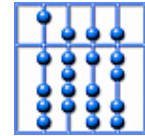
1. Introduction
2. Basics: Behavior, Interaction, Concurrency
3. Coroutines
4. Data Flow Models
  1. Synchronized Models
  2. Perfectly Synchronized Models
  3. Message-Asynchronous Models
5. Communicating Processes
6. State-Based Models
7. Coordination
8. Executions
9. Property Descriptions



## Questions

---

1. Exercise: Implement sender and receiver of the alternating bit protocol using automata and SDL.
2. Describe an (angelic) merge component using SDL. What are its observation traces?
3. Is a similar component implementable using Kahn functions? Explain your answer.
4. What is the consequence of this observation for programming languages with explicit communication constructs?
5. Define the observation traces for alarm clock process using timed observation traces. What are the equivalent untimed (structured) observation traces?
6. Do the structured observation traces of Question 5 describe a Kahn function?



## Exercise 1

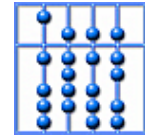
---

Implement sender and receiver of the alternating bit protocol using automata and SDL.



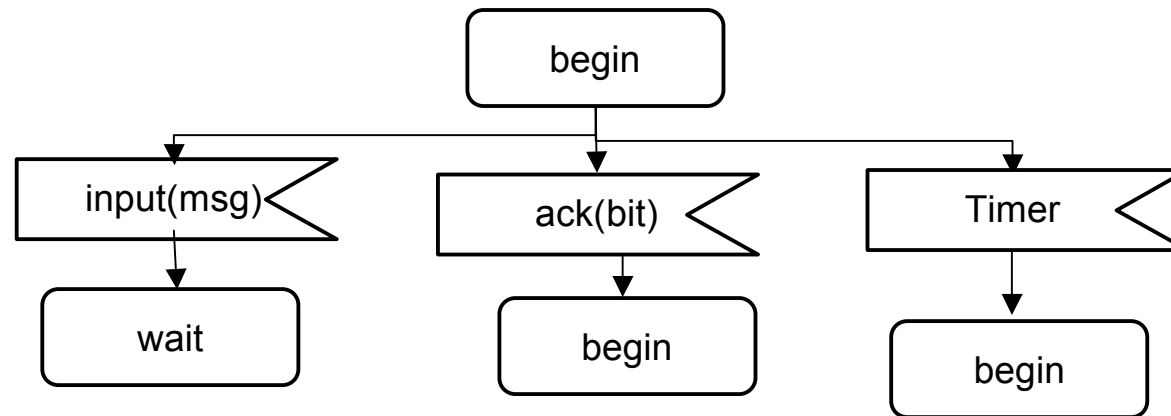
Alternating bit protocol: Reliable transmission over an unreliable (lossy) medium

- Sender: Re-transmit current input (including current bit flag) until acknowledgement (current bit flag) is received.
- Receiver: Output current body if a new message is received (new bit flag), echo current bit flag.
- Medium: Transmit current message or loose message.



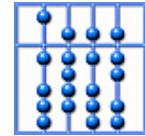
## Exercise 1

---

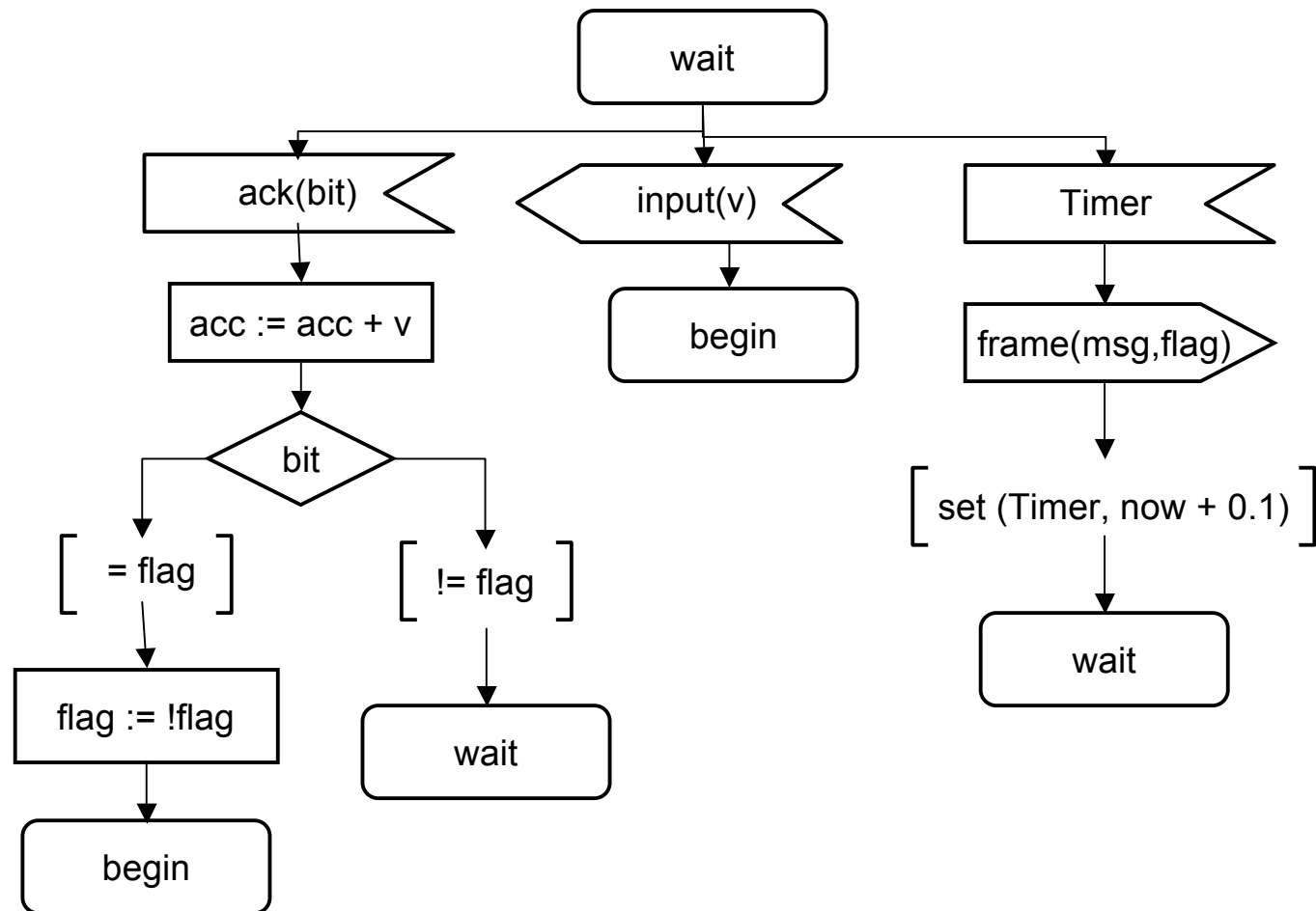


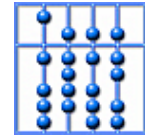
Sender: Two-phase protocol:

- Phase 1: Wait for input, ignore all other messages
- Phase 2: Cyclically retransmit input with current flag
  - Proceed to new input if acknowledgement was received
  - Save new input for later processing
  - Retransmit in case out timeout



# Exercise 1

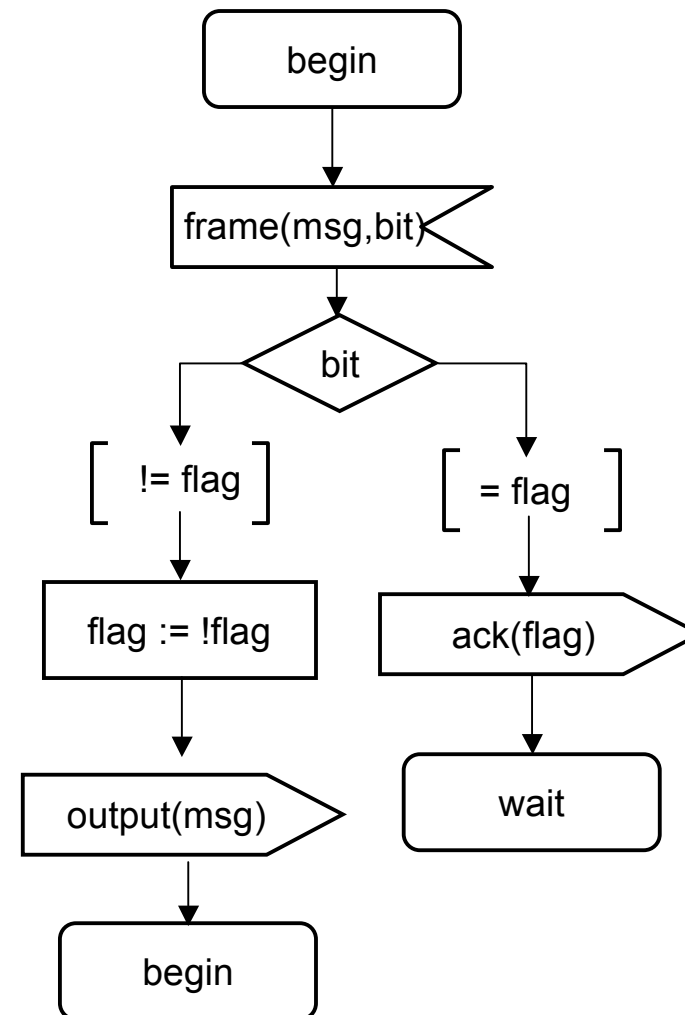


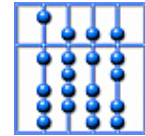


## Exercise 1

Receiver: One-phase protocol:

- Case 1 (Copy received):  
Retransmit flag
- Case 2 (New frame received):  
Forward output

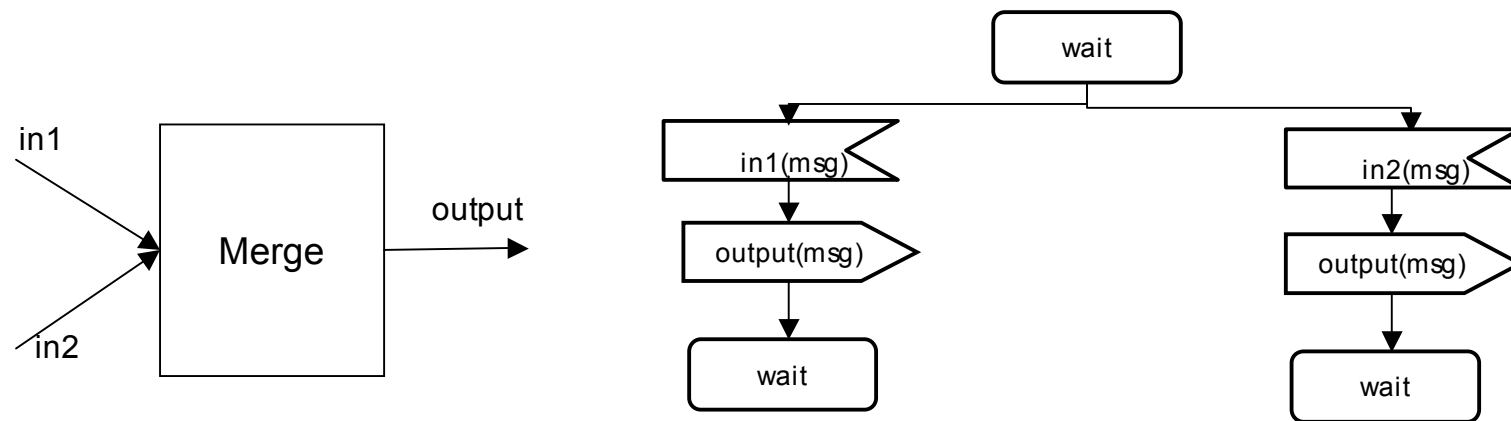


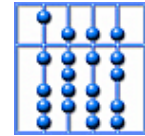


## Question 2

Describe an (angelic) merge component using SDL. What are its observation traces?

(Angelic) merge component: Wait for messages on input channels, forward these messages to output channels





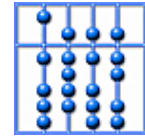
## Question 2

---

Describe an (angelic) merge component using SDL. What are its observation traces?

Observation traces:

- Examples:
  - $\langle \rangle$
  - $\text{in1}(1) \cdot \text{in1}(2) \cdot \text{out}(1) \cdot \text{out}(2)$
  - $\text{in1}(1) \cdot \text{in1}(2) \cdot \text{out}(1) \cdot \text{in2}(3) \cdot \text{in1}(4) \cdot \text{out}(2) \cdot \text{out}(3) \cdot \text{out}(4)$
- Formalization: Characterizing observation traces  $t$ 
  - No “bad” messages: Only received inputs are send as output:  
 $s \leq t \Rightarrow s \text{ © } (\text{In1} \cup \text{In2}) \leq s \text{ © } \text{Out}$
  - All “good” messages: All received input is forwarded eventually:  
 $t \text{ © } (\text{In1} \cup \text{In2}) = t \text{ © } \text{Out}$



### Question 3

---

Is a similar component implementable using Kahn functions?  
Explain your answer.

Kahn functions: Monotonic (continuous) functions on untimed observation traces.

Monotonic function  $F: s \leq s' \Rightarrow F(s) \leq F(s')$

Angelic merge:

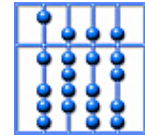
$\text{Merge}(\langle \rangle, b) = b$

$\text{Merge}(a, \langle \rangle) = a$

$(a, \langle \rangle) \leq (a, b) \Rightarrow \text{Merge}(a, \langle \rangle) = a \leq \text{Merge}(a, b)$

$(\langle \rangle, b) \leq (a, b) \Rightarrow \text{Merge}(\langle \rangle, b) = b \leq \text{Merge}(a, b)$

Therefore: The angelic merge cannot be implemented using Kahn functions



## Question 4

---

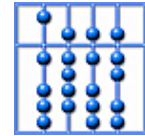
What is the consequence of this observation for programming languages with explicit communication constructs?

Programming languages with explicit communication constructs generally use blocking input statements (e.g., waiting on a non-empty input buffer).

Sequential programming languages generally do not support simultaneous waiting on several blocking constructs (e.g., waiting on shared resources).

To implement (untimed) components like a merge component in a data flow like fashion,

- either timed constructs (e.g., timeouts) must be added to the language
- or language constructs for the simultaneous waiting on input (e.g., occam alt construct, SDL input choice)



## Question 5

---

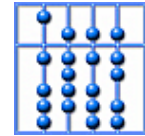
Define the observation traces for the alarm clock process using timed observation traces. What are the equivalent untimed (structured) observation traces?

Alarm clock:

- Once started, clock will run till stopped or alarm raised
- Stopping a alarm that has gone of has no effect

Timed observation traces:

- $\langle \rangle$
- $(\text{start}, -) \cdot (-, -) \cdot \dots \cdot (-, -) \cdot (-, \text{ring}) \cdot (-, -) \cdot \dots$
- $(\text{start}, -) \cdot (-, -) \cdot \dots \cdot (-, -) \cdot (-, \text{ring}) \cdot (\text{stop}, -) \cdot (-, -) \cdot \dots$
- $(\text{start}, -) \cdot (-, -) \cdot \dots \cdot (\text{stop}, -)$



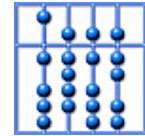
## Question 5

---

Define the observation traces for the alarm clock process using timed observation traces. What are the equivalent untimed (structured) observation traces?

Untimed observation traces:

- Unstructured examples:
  - $\langle \rangle$
  - $\text{start} \cdot \text{ring}$
  - $\text{start} \cdot \text{ring} \cdot \text{stop}$
  - $\text{start} \cdot \text{stop}$
- Structured examples:
  - $(\langle \rangle, \langle \rangle)$
  - $(\text{start}, \text{ring})$
  - $(\text{start} \cdot \text{stop}, \text{ring})$
  - $(\text{start} \cdot \text{stop}, \langle \rangle)$



## Question 6

---

Do the structured observation traces of Question 5 describe a Kahn function?

Examples for structured observation traces (assuming visible “ticks”):

- (start,ring)
- (start • stop, ring)
- (start • stop, <>)

“Functional” interpretation:

- Alarm(start) = ring
- Alarm(start • stop) = ring
- Alarm(start • stop) = <>

Since Alarm(start • stop) = ring  $\neq$  <> = Alarm(start • stop) no functional interpretation for this abstraction from time