Website: http://concurrency.cs.uni-kl.de/teaching.html

↳ Information about the course

↳ Exercise sheets

Structure of the lecture:

Multithreaded programs (Part I) → State networks (Part II) → Dynamic networks (Part III)

Part II: State Networks

Goal: Verify network protocols

Examples:
- Alternating bit protocol (this course)
- Sliding window protocols (more general, layer 2 of OSI 7 layer model)

Automaton model:
- Protocols of interest are designed to operate correctly in the presence of unreliable channels
  - ↳ Finite state automata model full behaviour of system
  - ↳ Automata communicate via unbounded and lossy FIFO channels

Unbounded = infinite state model

Lossy = gives decidability

Exercise: Show the halting problem is decidable for lossy Turing machines.

Verification problems:
- ↳ Safety properties: Is an undesirable state reachable?
  - This is again infinite-state decidability
- ↳ Eventuality properties: Do all computations lead to some set of states.

Example: Alternating Bit Protocol

Sender:

- Get msg to be sent
- Append sequence number
- Send over channel M to receiver

- Await ack from receiver with same sequence nbr

- If no ack arrives within some time, retransmit

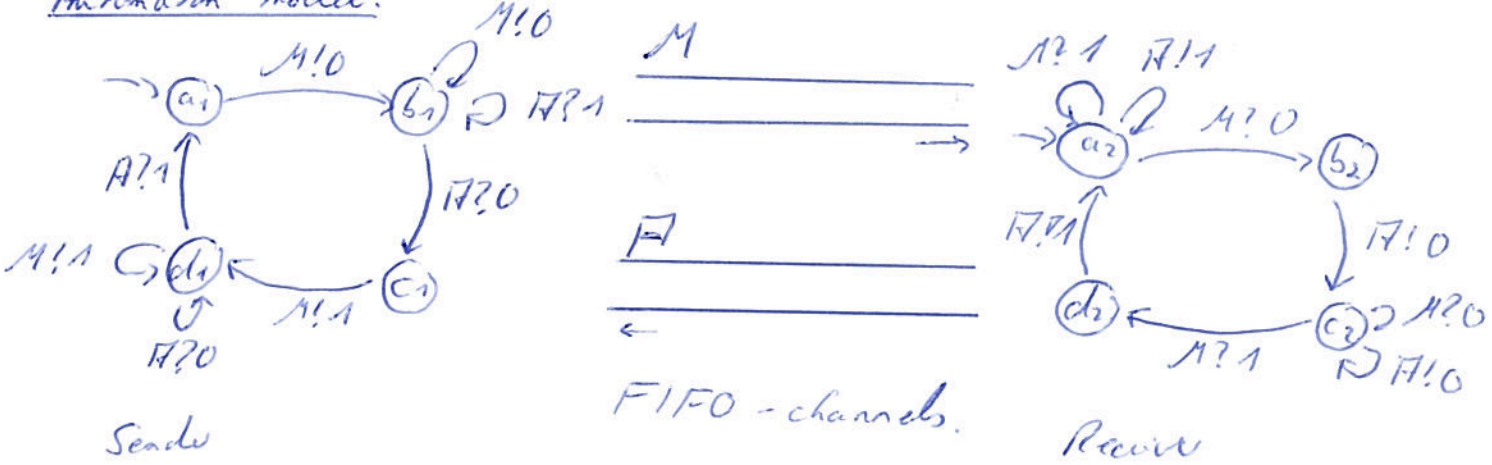
- When ack has arrived, repeat with next msg, but sequence number inverted.

Receiver:

- Receive msg and sequence number over channel M
- If correct, deliver to system and await msg with inverted number (b_2)
- Discard msg with unexpected number
- Receiver sends ack over A with number of last msg received.

> Note that msg content unimportant for the protocol.

Automaton model:



Protocol transmits msgs from sender to receiver in correct order

- Although channels may lose msgs
- May corruption could also be modelled as loss
- Channels do not reorder messages

A little bit of semantics of LCS:

Configurations are pairs $\gamma = (s, w)$

- s = control states
- w = channel contents, $w(M)$, $w(A)$ are words.

- Transitions are defined by the control part + msg loss:
  - ↳ Sending $a_1 \xrightarrow{M!O} b_1$ appends msg O to end of channel M

    $((a_1, a_2), \underset{M}{\varepsilon}, \underset{A}{\varepsilon}) \to ((b_1, a_2), O, \varepsilon) \to ((b_1, a_2), O\underline{O}, \varepsilon) \ldots$
  - ↳ Receiving $a_2 \xrightarrow{M?O} b_2$ removes msg O from head of channel M:

    $((b_1, a_2), \underline{O}O, \varepsilon) \to ((b_1, b_2), O, \varepsilon)$

    Executing a send of an acknowledge:

    $((b_1, b_2), O, \varepsilon) \to ((b_1, c_2), O, O)$.
  - ↳ Any msg can be lost:

    $((b_1, c_2), O, O) \to ((b_1, c_2), O, \varepsilon)$ when trying to send another acknowledge.

Reachability problem

Given: LCS L and (bad) configuration γ.
Question: Is γ reachable from the initial configuration γ_0 of L?

Key trick:

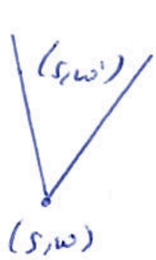
Configurations are ordered:

$(s_1, w_1) \leq (s_2, w_2)$ if $s_1 = s_2$ and $w_1(C)$ is a subword of $w_2(C)$ f.a. C.

x b is a subword of a x y b z

Backwards search:

- ↳ Search γ_0 backwards, starting from γ.
- ↳ Define backwards transition relation $\gamma_1 \leadsto \gamma_2$
  - => Carefully handle msg. loss (*)
- ↳ Search is a priori not bounded
  - => ≤ - minimal elements sufficient
  - => There are finitely many according to Higman.



* Intuition:
A configuration (s,w) represents all configurations (s,w') that are larger. $(s,w) \leq (s,w')$

Instance of
reachability problem: Is $((b_1, c_2), \varepsilon, 0)$ reachable from $((a_1, a_2), \varepsilon, \varepsilon)$ in RBP?

Init: TODO = $\{((b_1, c_2), \varepsilon, 0)\}$
DONE = $\emptyset$

Pre 1: TODO = $\{((a_1, c_2), \varepsilon, 0), ((b_1, c_2), \varepsilon, 0), ((b_1, c_2), 1.0),$
$((b_1, b_2), \varepsilon, \varepsilon), ((b_1, c_2), 0, 0)$ $((b_1, c_2), \varepsilon, \varepsilon)\}$

(annotations: "Sender just received !1=1"; "undo sending of !1:0"; "We have $((b_1, c_2), 0.0) \geq ((b_1, c_2), \varepsilon, 0)$.")

DONE = $\{((b_1, c_2), \varepsilon, 0)\}$

Pre 2: TODO = $\{((a_1, c_2), \varepsilon, 0), ((b_1, c_2), \varepsilon, \varepsilon), ((a_1, b_2), \varepsilon, \varepsilon),$
$((b_1, a_2), 0, \varepsilon))\}$
DONE = $\{((b_1, c_2), \varepsilon, 0), ((b_1, b_2), \varepsilon, \varepsilon)\}$

Pre 3: TODO = $\{((a_1, a_2), \varepsilon, \varepsilon) \ldots$
return reachable.

General remarks:

Well-quasi-orderings:
Termination of above algorithm relies on wqo-theory
↳ Every infinite sequence of configurations $\gamma_1, \gamma_2, \ldots$
contains two comparable elements $i < j$ with $\gamma_i \leq \gamma_j$.
(Try out sequences in $\mathbb{N}$ or $\mathbb{N}^2$)

Symbolic representations:

- Infinite sets of configurations represented by symbolic configurations
$((a_1, b_2), (0+1)^* . (0+\varepsilon), (1+\varepsilon) . (0+1)^*)$
- Theory of downward-closed regular languages.
- Acceleration: Compute effect of control loops in the limit.

Part III: Dynamic networks

Goal:
- Verify dynamic network protocols
  - ↳ Process and links generated / destroyed at runtime (dynamically)
  - ↳ Synchronous (handshake) communication (no buffers / queues)
- Interested in connection properties:
  - ↳ PAN / LCS: mutex
  - ↳ Hue's linkage
- Investigate "verifiability"
  - ↳ Restrictions necessary to obtain decidability of reachability / coverability / termination. (infinite state version of reachability)

Technically:
- Finite state programs
- unboundedly many instances / arbitrarily many connections
- → infinite state.

Model:

Pi-Calculus ≅ regular expressions enriched by | and νa

Example: Home banking

$$\nu ip.(L_{wl}, ip_1 \mid S_{wl})$$

where

$$C(u,i) := \overline{u}\langle i \rangle.\, i(s).\, \overline{s}\langle s \rangle.\, (L_{u,i})$$

$$S(u) := u(x).\, (\nu ses.\, \overline{x}\langle ses \rangle.\, ses(y).\, 0 \mid S_{wl})$$

Transitions:

unfold definitions →²

$$\nu ip.\, (L_{wl}, ip_1 \mid S_{wl})$$

$$\nu ip.\, \overline{wl}\langle ip \rangle.\, ip(s).\, \overline{s}\langle s \rangle.\, (L_{wl}, ip_1) \mid wl(x).\, (\nu ses.\, \overline{x}\langle ses \rangle.\, ses(y).\, 0 \mid S_{wl})$$

(a) $\rightarrow \nu ip. (ip(s). \bar{s}\langle s\rangle. (L\langle url, ip\rangle \mid \nu ses. \overline{ip}\langle ses\rangle. ses(y). 0)) \mid S\langle url\rangle$

(b) $\rightarrow$

$\nu ses. (\nu ip. \overline{ses}\langle ses\rangle. (L\langle url, ip\rangle \mid ses(y). 0)) \mid S\langle url\rangle$

(c) $\rightarrow \nu ip. (L\langle url, ip\rangle \mid S\langle url\rangle)$

Protocol:

(a) Client sends ip-address to server

(a) Server spawns new thread to handle client request.
Server available again for next client

(b) Thread sends session to client to establish private connection.
Uses ip-address previously received

(c) Session terminates at some point
↳ Client sends on session channel to thread
the session itself

Graphically