Tik-79.231 Parallel and Distributed Digital Systems

Lecturer: Esa Kettunen, M.Sc.
Esa.Kettunen@hut.fi
050-380 7284

Assistant: Jukka Honkola
Jukka.Honkola@hut.fi
09-451 5244

Requirements: Examination
Compulsory homework

Prerequisites: Tik-79.144, Tik-79.148, or equivalent

©Johan Lilius 1997, Keijo Heljanko 1998, Esa Kettunen 1999, 2000

Literature:	*Johan Lilius, Keijo Heljanko*, *Esa Kettunen*, Compendium (lecture slides, mostly) *Mikko Tiusanen*, Petri-verkkojen perusteet (in Finnish) *Kimmo Varpaaniemi*, PROD Reference Manual *David Walker*, Introduction to a Calculus of Communicating Systems,
Additional material:	Tadao Murata, *Petri Nets: Properties, Analysis and Applications* Robin Milner, *Communication and Concurrency*

©Johan Lilius 1997, Keijo Heljanko 1998, Esa Kettunen 1999, 2000

More information

- WWW-pages `http://www.tcs.hut.fi/Teaching/Tik-79.231.html`
  - Course brochure
  - Schedule
  - Practical work (i.e. tutorials given by the assistant)
  - Home assignments
  - Links
- Otax news group: opinnot.tik.rhj
  - any urgent information

©Johan Lilius 1997, Keijo Heljanko 1998, Esa Kettunen 1999, 2000

Goal of the course

Accumulation of some basic skills in the specification, verification, and validation of concurrent and distributed systems

by learning the following techniques:

- basics of Petri nets, temporal logics, (and process algebra), and
- modelling and analysis of systems.

©Johan Lilius 1997, Keijo Heljanko 1998, Esa Kettunen 1999, 2000

Other related courses

- Tik-79.179 Rinnakkaiset ja hajautetut digitaaliset järjestelmät
- Tik-79.186 Reaktiiviset järjestelmät
- Tik-79.190 Rinnakkaisohjelmistojen testaus
- Tik-79.193 Rinnakkaisohjelmistojen määrittelymenetelmät
- Tik-76.164 Sulautetut järjestelmät
- Tik-76.166 Rinnakkaisohjelmointi
- Tik-79.146 Logiikka tietotekniikassa: Erityiskysymyksiä I
- (Tik-79.185 Verifionti)

©Johan Lilius 1997, Keijo Heljanko 1998, Esa Kettunen 1999, 2000

Concurrent and distributed systems

- Concurrency: when two event sequences are executed in parallel, two events are *concurrent* if they coincide. The event sequences are executed *concurrently*.
- Distribution: there is no centralized control in a concurrent system.

  A distributed system is a system where your own machine is broken down by a malfunction of a machine you did not even know existed (L. Lamport)
- Asynchronicity: different subsystems are not paced (synchronized) with each other (so they can have different speeds)
- Reactivity: the system is continuously working and it is controlled by outside stimuli (i.e., signals) to which it reacts.

©Johan Lilius 1997, Keijo Heljanko 1998, Esa Kettunen 1999, 2000

Parallel and distributed system

Design problems:

- Large number of states due to high level of nondeterminism
- Dependencies between the subsystems are hard to control and understand

$\Rightarrow$ it is difficult to verify/validate wanted properties.

Why bother with parallel and distributed systems?

- Increase of power
- Redundancy
- Modularity
- Physical/geographic distance between pieces of hardware

©Johan Lilius 1997, Keijo Heljanko 1998, Esa Kettunen 1999, 2000

Properties

- Security properties:
  "Nothing bad ever happens"
  - Deadlock-freedom
  - Mutual exclusion
  - Data security
  - Certainty that the system never goes to some (unwanted) state
- Liveness porperties:
  "Something good eventually happens"
  - Fairness
  - Guaranteed signal delivery
  - Guaranteed service for each service request
- Others:
  - P occurs infinitely often
  - After P, Q will occur after n steps/seconds
  - In each state C is satisfied

©Johan Lilius 1997, Keijo Heljanko 1998, Esa Kettunen 1999, 2000

Methods taught in this course

- Petri nets
- Temporal logics
- PROD
- Process algebras
- Equivalences

©Johan Lilius 1997, Keijo Heljanko 1998, Esa Kettunen 1999, 2000

Application of methods

- Telecommunications
  - Protocol verification
  - Protocol testing
  - Performance analysis
- Safety critical systems
  - Guidance logics of railroad systems
  - Logics of aeroplane piloting systems
  - Flight control systems
- Hardware design
  - Microprocessors
- Reactive systems & embedded systems in general

©Johan Lilius 1997, Keijo Heljanko 1998, Esa Kettunen 1999, 2000

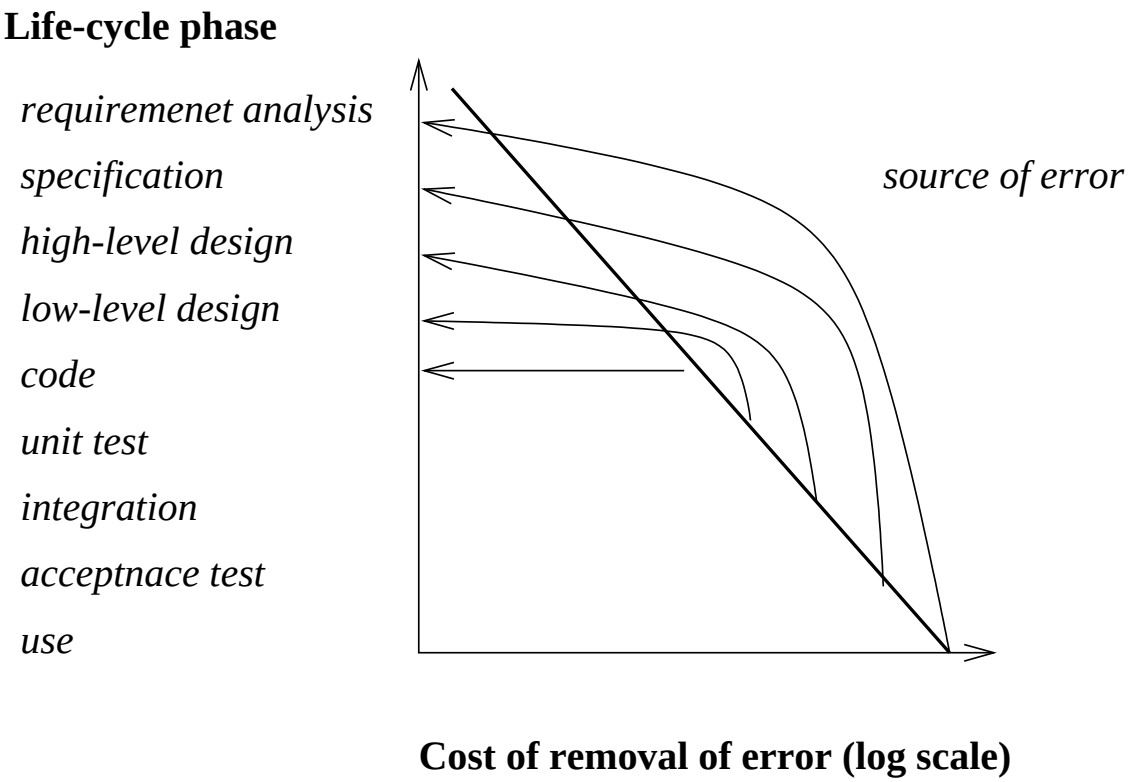
Software crisis

- How to design and implement software while guaranteeing:
  - Timely delivery,
  - No budgetary overflows,
  - 100 % absence of bugs at time of delivery,
  - Adherence to customer's original requirements,
  - Ease of expandability,
  - ...
- Writing software is more or less done by hand/brain/people
- Software Engineering
- Formal Methods are an attempt to solve some of the above mentioned problems.

©Johan Lilius 1997, Keijo Heljanko 1998, Esa Kettunen 1999, 2000

Software crisis

- Size of software increases -> price increases -> bugs/errors are more and more expensive
  - Bug in AT&T switching software: half of U.S. telephone network down
  - Ariane 5 exploded because of a software malfunction
  - Bug in Pentium FPDIV



©Johan Lilius 1997, Keijo Heljanko 1998, Esa Kettunen 1999, 2000

What is specification?

- Specification is the description if the assumptions and requirements about the functioning of the system fulfilling the following criteria:
  - Unambiguousness
  - Not necessarily a complete specification
- Specification is a description of allowable computations
  - Computation is understood here as widely as possible
- Specification can be interpreted as an AGREEMENT between client and vendor/manufacturer as what the software should do.
- Typically, specification means specification with some formal method.

©Johan Lilius 1997, Keijo Heljanko 1998, Esa Kettunen 1999, 2000

Advantages of specification

- Correctness: the specification is checked by automated tools
  - Validation
  - Verification
- Completeness: the specification works as a notebook
- Consistency: the specification helps detect contradictory and unreasonable requirements early
- Reusability: Software Factory.
- Formal description technique, specification method = specification + validation + verification
- SA, ER, different OO methods (OMT, UML), etc. are not specification methods in the same sense

©Johan Lilius 1997, Keijo Heljanko 1998, Esa Kettunen 1999, 2000

Specification as part of the Software Engineering Process

- Requirements analysis
  - Specification method clarifies customer's requirements
- System design
  - Several methods to support decomposition and refinement
- System verification
  - Verification is impossible without formal methods
- System validation
  - It is possible to derive a set of tests from the specification with which the system can be validated
- System documentation
  - Specification interpreted as documentation
- System analysis and evaluation
  - Often specification methods are applied in modelling and verification of already existing systems: reverse engineering

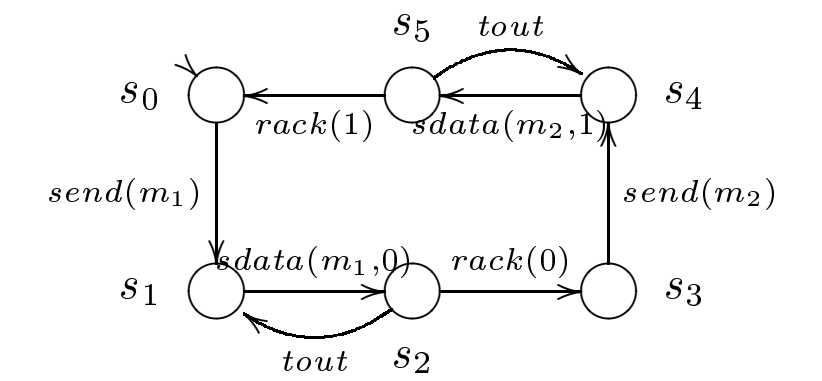
©Johan Lilius 1997, Keijo Heljanko 1998, Esa Kettunen 1999, 2000

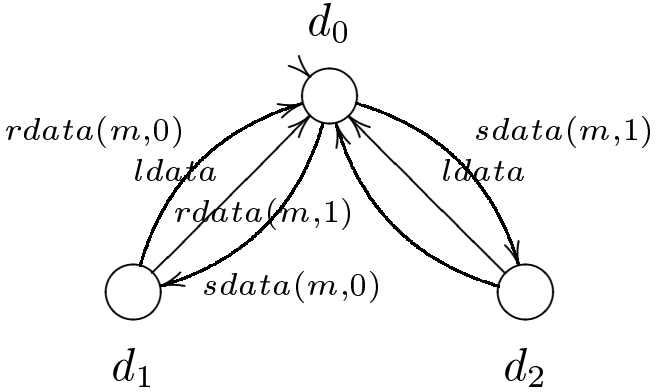
Example: Alternating Bit Protocol

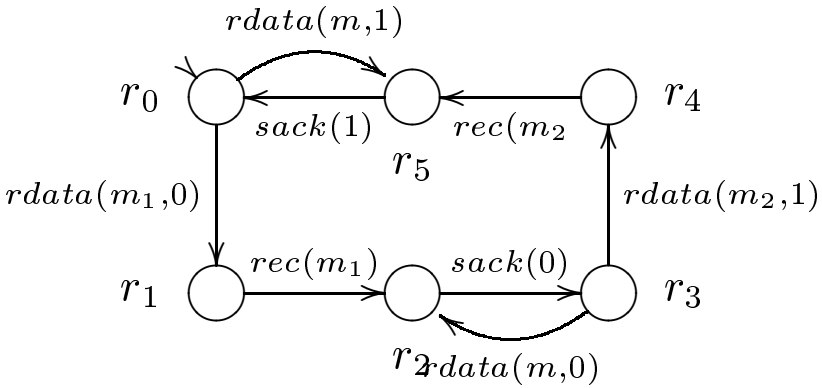
- AB protocol is a simple telecommunication protocol
- Messages are sent tagged with bits 0 or 1.
  - Sender process: once started, the message is sent with bit b.
    - If the message gets lost the process timeouts and the message is resent with bit b.
    - If the process receives bit b as a handshake then it will send a new message with bit $\hat{b} = 1 - b$.
  - Receiving process: process receives message with bit b. Bit b is acknowledged as a handshake.
    - If the process receives an incorrect bit it will be forgotten.
    - If the process receives a new message with bit $\hat{b}$ this will be acknowledged.
  - Ergo: Protocol sends and receives messages alternating the tagged bit.
- Claim: Protocol can recover if the message is lost at most a finite number of times.

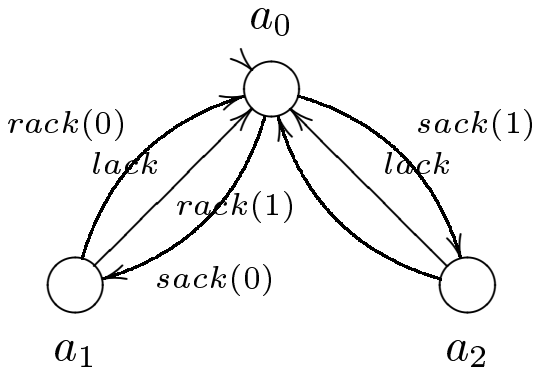
©Johan Lilius 1997, Keijo Heljanko 1998, Esa Kettunen 1999, 2000

Labelled transition systems (communicating automata)







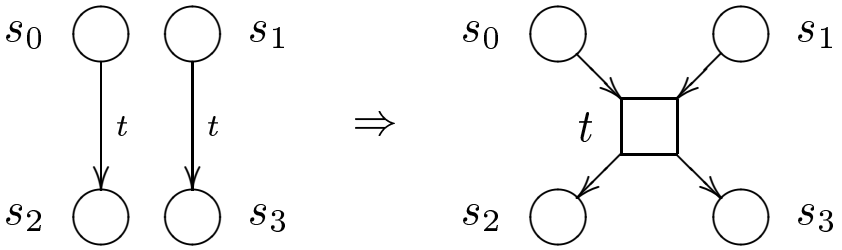


- Each subsystem has its own state! Initial state is marked with $>$.
- Events with the same name occur at the same time (communication, synchronization).

©Johan Lilius 1997, Keijo Heljanko 1998, Esa Kettunen 1999, 2000

Description as a Petri Net

- *Granularity* of concurrency is a problem for labelled transition systems.
  - Automata are sequential.
  - Concurrency exists only between automata.
- Petri nets can be seen as a generalization of labelled transition systems.
- Basic idea: Describe synchronization explicitly.

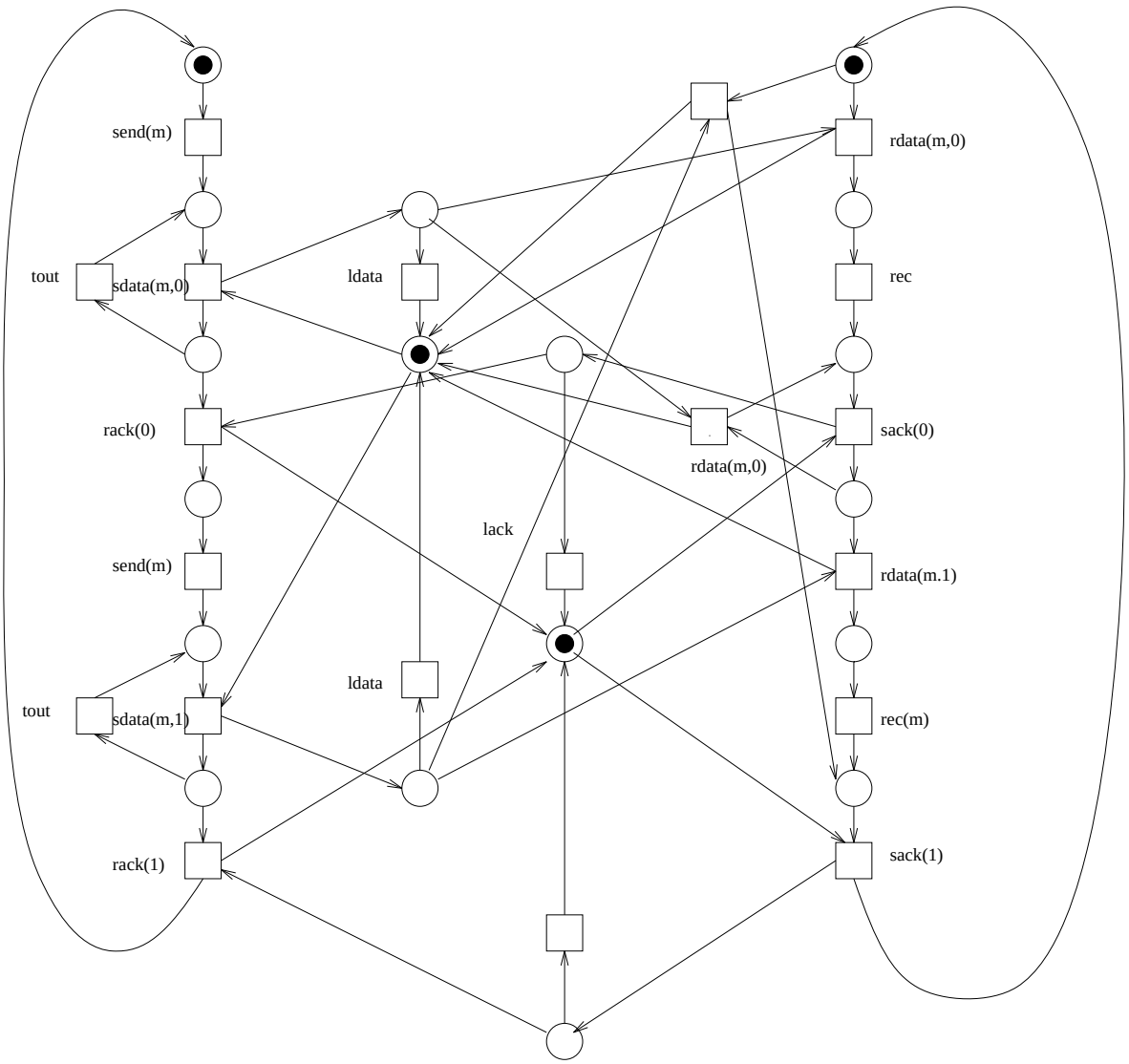


Basic elements of a Petri net: Petri-verkon (perus-)osat:

- Place: p ◯
- Transition: t □
- Directed arc: ⟶
- In a Petri net the following holds:
  - The global state of the system consists of several local states.
  - The state of the system is *distributed*.
  - The subsystems work *in parallel*.

©Johan Lilius 1997, Keijo Heljanko 1998, Esa Kettunen 1999, 2000

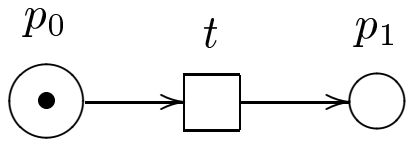
Description as a Petri net



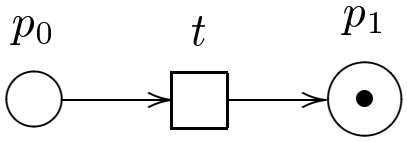
©Johan Lilius 1997, Keijo Heljanko 1998, Esa Kettunen 1999, 2000

Token game

- Describe the system state by inserting tokens (black dots) into the places.
- The system makes a state change by firing transitions.
- A transition is fired if all of its predecessor places have a token and all of its successor places are empty (*there are going to be modifications in the firing condition later*)
- The new state of the system is reached by removing the tokens from the predecessor places and inserting tokens to each of the successor places.
- t Concession:



- t Firing:



©Johan Lilius 1997, Keijo Heljanko 1998, Esa Kettunen 1999, 2000

Token game

- Behaviour of the system can be described by sequences of state transitions.
  - E.g., successful sending of a message: $send(m) \rightarrow sdata(m,0) \rightarrow rdata(m,0) \rightarrow rec(m) \rightarrow sack(0) \rightarrow rack(0) \rightarrow send(m) \ldots$
- The state transition sequences are part of a directed graph which describes all the possible state transitions in the system.
  - State space: all the possible states of a system.
  - Reachability graph: directed graph whose node are states and whose arcs are state transitions.
- By playing the token game by hand it 'fairly' easy to validate the AB protocol.

©Johan Lilius 1997, Keijo Heljanko 1998, Esa Kettunen 1999, 2000