1 PROCESS ALGEBRA

- In the sequel we will be describing an alternative to Petri nets in the specification of parallel (concurrent) and distributed digital systems.
- Process algebra is the common term for a set of specification methods for parallel and distributed systems. In this course the process algebra will be CCS (Calculus of Communicating Systems). Other process algebras are, e.g., CSP, ACP, and LOTOS.
- An algebra is a mathematical structure consisting of elements, operations, and equations between elements and operations.
- A process is a set of events.
- This means that a process algebra comprises events and operations to combine events.

©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997–2000

Introduction

- A Petri net describes causal dependencies of a system.
- In a process algrebra our premises are different:
  - A sequential program can be understood as a mathematical function describing one state of a memory to another. An underlying assumption in this so called functional approach is that every program itself controls its memory space.
  - If we allow parallel programs, sharing the memory, the situation is fundamentally changed.

    (1) `X:=1` (2) `X:=0;X:=X+1`
  - If the programs work fully independently, their effect is the same.
  - However, if there exists another program which is able to write `X:=1` at any moment, programs (1) and (2) no longer have the same effect.

©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997–2000

Introduction

- A central primitive of process algebra os

  COMMUNICATION.

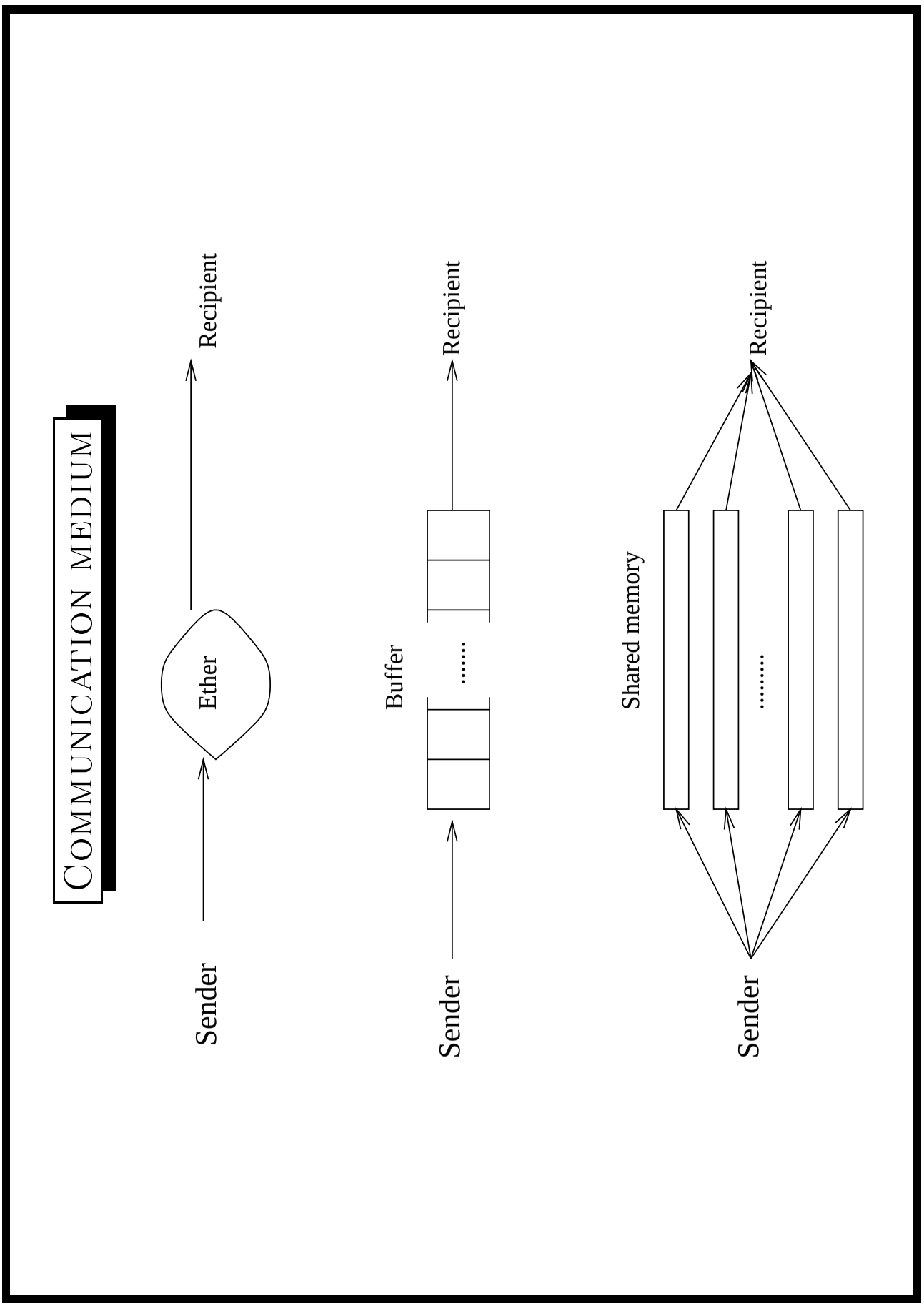
  Therefore communication and interaction between systems or subsystems is studied.
- In process algebra there are operations to join communication events together.
- One of the shortcomings of Petri nets is the lack of operations with which nets can be joined. (*This is not quite true – joining nets can be done but is not quite as 'natural', however, as joining of events in process algebras.*)

©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997–2000

Communication event

- Communication and concurrency support conceptually each other:
  - A system has *diversity*: It is built of different parts functioning in parallel.
  - A system has *unity*: Its different parts communicate with each other.
- Each part of a system has an identity, it is an *AGENT*.
- The behaviour of a system is studied by observing its events.
- The events are observed by identifying the agents taking part in them.
- An can take part in an event with other agents, thus making it a *COMMUNICATION EVENT*, or
- The function of an agent may be independent of the functionings of other agents, thus creating a *PARALLEL EVENT*.
- Actually, the assumption can be made that such an independent functioning consists of internal communication events of the agent.
- It is essential to make a difference between visible or *EXTERNAL* communication events and invisible or *INTERNAL* communication events.

©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997–2000



©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997–2000

Communication medium

- What is in common with different communication events?
  - Sender.
  - Medium,
  - Recipient,
  - Communication is message based.
- The sending of a message, its transmission via a medium, and its receipt can be understood as a single time-granular event.
- The medium can be understood as a participant, much like the agent.
- In process algebra, there is no special treatment for the medium. If there is a need to model the medium, it can be modelled as an agent.
- Hence, communication is a granular event, experienced by (at least) two agents. This kind of an event is also called a handshake.

$$S \longrightarrow R$$

©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997–2000

Correctness of a system

- How to ascertain the (correct) functioning of a system?
- The behaviour of a system is what can be observed.
- Observing a system necessitates communication with it.
- This means a system function correctly if its external communication events match the communication events in the specification.
- This kind of a notion correctness is called *observational equivalence*.

©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997–2000

CCS: Syntax

- Defined is a syntax with which agents can be described.
- Agents model processes.
- Agents communicate via ports.
- Part can be either input or output ports.
- Each port has a label.
- The label of an outgoing port has a line above it (e.g., $\overline{in}$).
- Let us take as given a set $\mathcal{L}$ of label comprising of a set of input labels $\mathcal{I}$ and output labels $\mathcal{O}$. Let l, k be arbitrary labels. If l is a label then its complement is $\bar{l}$.

©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997–2000

Syntax

Nil

- The agent *Nil* does nothing.
- The agent *Nil* is used denote the termination of a process.

Prefixing

- If l is a label and P is an agent, $l.P$ is the agent whose corresponding process is able to perform the function l first and then the functions of P.
- E.g., a match can be lighted and then it can do nothing:: $Light.Nil$.

©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997–2000

Syntax

Definition

- An agent can be defined with a definition statement:
  $Match \stackrel{\text{def}}{=} Light.Nil$
- A definition can also be recursive:
  $Clock \stackrel{\text{def}}{=} tick.Clock$
- An individual event is able have a parameter:
  $Addone \stackrel{\text{def}}{=} in(x).\overline{out}(x+1).Addone$
- An agent can be defined with several statements:
  $Addone \stackrel{\text{def}}{=} in(x).Holding(x)$
  $Holding(y) \stackrel{\text{def}}{=} \overline{out}(y+1).Addone$

©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997–2000

Syntax

Sum

- The following statement represents a machine which changes 50 pence to two coins of 20 pence an single coin of 10 pence:
  $Change \stackrel{\text{def}}{=} fifty.\overline{out}(20).\overline{out}(20).\overline{out}(10).Change$
- Let us extend the machine so that it changes a 1 mark coin to two coins of 50 pence:
  $Change1 \stackrel{\text{def}}{=} hundred.\overline{out}(50).\overline{out}(50).Change1$
- The functioning of the new machine can be described with the sum operation:

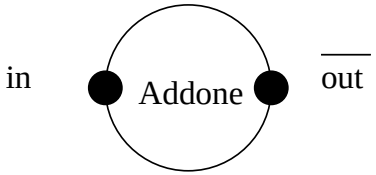
$$\begin{aligned} NewChange \stackrel{\text{def}}{=} & \quad fifty.\overline{out}(20).\overline{out}(20).\overline{out}(10).NewChange \\ + & \quad hundred.\overline{out}(50).\overline{out}(50).NewChange \end{aligned}$$

©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997–2000

SYNTAX

Communication and parallel composition

- It is easier to describe inter-process communication with the help of a picture:



representing the agent $Addone$.

©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997–2000

Syntax

- The agent
  $Double \stackrel{\text{def}}{=} rec(x).\overline{in}(2x).Double$
  is depicted by the picture

©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997–2000

SYNTAX

- Parallel composition of two agents is described with the operator $|$. Hence, $P|Q$ is an agent where P and Q can function independently and communicate at will.
- The agent $DA \stackrel{\text{def}}{=} Double|Addone$ is depicted by the following picture:

©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997–2000

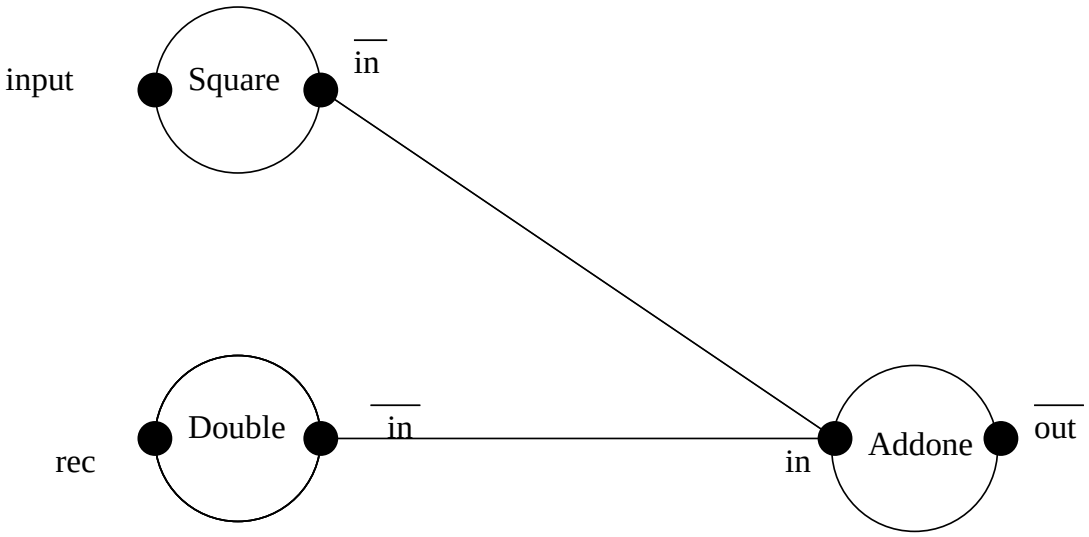
Syntax

- DA can be joined with agent

$$Square \stackrel{\text{def}}{=} input(y).\overline{in}(y^2).Square$$

resulting

$$SDA \stackrel{\text{def}}{=} Square|DA$$



- Problem: So far all labels are global.

©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997–2000

Syntax

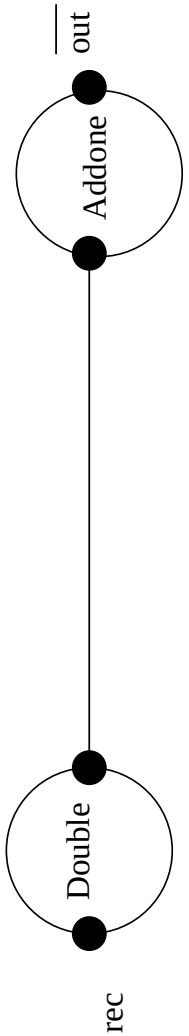
Restriction

- There is often a need to restrict the visibility of a label.
- For this purpose, the restriction operator $\backslash$ exists.
- $P\backslash l$ is an agent similar with P but unable to communicate via the ports $l, \bar{l}$.

©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997–2000

SYNTAX

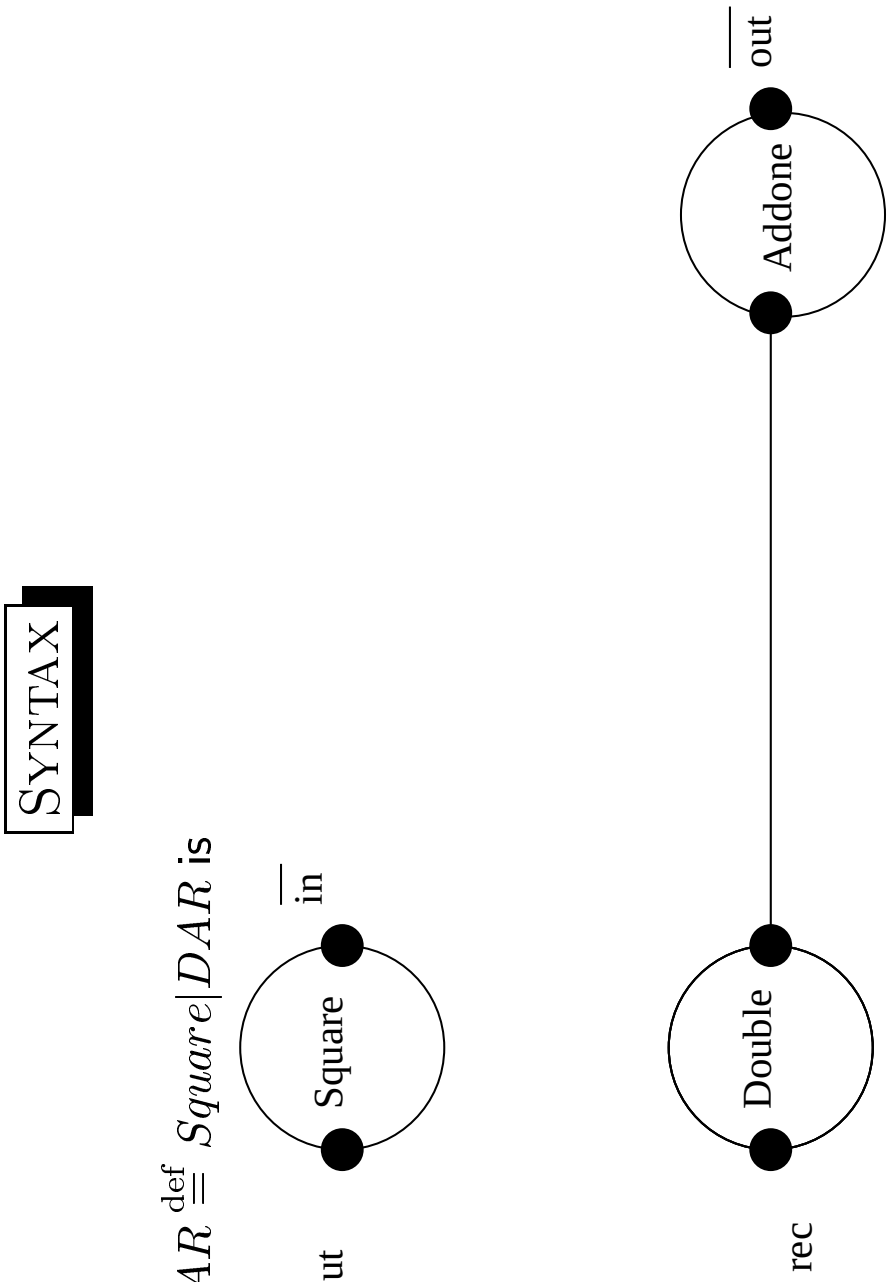
- Eg., $DAR \stackrel{\text{def}}{=} DA \backslash in$



©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997–2000

SYNTAX

- Now $SDAR \stackrel{\text{def}}{=} Square|DAR$ is



©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997–2000

SYNTAX

Choice

- Given are the following agents:

$$
\begin{aligned}
User1 &\stackrel{\text{def}}{=} in1(x).(\overline{send1}(x).rec1(y).\overline{out1}(y).User.1 \\
&\quad + \overline{send2(x)}.rec2(y).\overline{out1}(y).User1) \\
User2 &\stackrel{\text{def}}{=} in2(x).\overline{send2}(x).rec2(y).\overline{out2}(y).User2 \\
Test1 &\stackrel{\text{def}}{=} send1(z).if\ \ z = 0\ \ then\ \ \overline{rec1}(\mathbf{T}).Test1\ \ else\ \ \overline{rec1}(\mathbf{F}).Test1 \\
Test2 &\stackrel{\text{def}}{=} send2(z).if\ \ z = 0\ \ then\ \ \overline{rec2}(\mathbf{T}).Test2\ \ else\ \ \overline{rec2}(\mathbf{F}).Test2
\end{aligned}
$$

- The Agent $\text{if } e \text{ then } P \text{ else } Q$ functions like P if e is true and like Q if e is false.
- Using these agents, the following systems can be constructed:

$$
\begin{aligned}
SystemA &\stackrel{\text{def}}{=} ((User2 \mid Test2) \mid (User1 \mid Test1))\backslash\{send1, send2, rec1, rec2\} \\
SystemB &\stackrel{\text{def}}{=} ((User2 \mid Test2)\backslash\{send2, rec2\}) \mid ((User1 \mid Test1)\backslash\{send1, rec1\})
\end{aligned}
$$

©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997–2000

SYNTAX

Renaming

- Given is a function $f : \mathcal{L} \to \mathcal{L}$. This kind of a function is called a renaming function.
- $P[f]$ is an agent functioning like P but its communication via the port renamed by function f.
- Consider the following agent:
  $Addone \stackrel{\text{def}}{=} in(x).\overline{out}(x+1).Addone$
  and its functions f_1 and f_2
  $f_1(out) = mid, f_1(\overline{out}) = \overline{mid}, f_1(l) = l$ muuten,
  $f_2(in) = mid, f_2(\overline{in}) = \overline{mid}, f_2(l) = l$ muuten.

©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997–2000

Syntax

- Now we can define
  $Addone1 \stackrel{\text{def}}{=} Addone[f_1]$ and
  $Addone2 \stackrel{\text{def}}{=} Addone[f_2]$.
  A conventional notation is:
  $Addone1 \stackrel{\text{def}}{=} Addone[\overline{mid}/\overline{out}]$ and
  $Addone2 \stackrel{\text{def}}{=} Addone[mid/in]$.
- Hence, an agen can be created:
  $Addtwo \stackrel{\text{def}}{=} (Addone1|Addone2)\backslash mid$.

©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997–2000

Summary

- The syntax of CCS has the following features:
- Agent: Nil.
- Prefixing: $l.P$.
- Sum: $P + Q$.
- Composition: $P|Q$.
- Restriction: $P\backslash L$.
- Renaming: $P[f]$.
- Choice: $\text{if } e \text{ then } P \text{ else } Q$
- Rekursive defintion: $agentti \stackrel{\text{def}}{=} lauseke$.

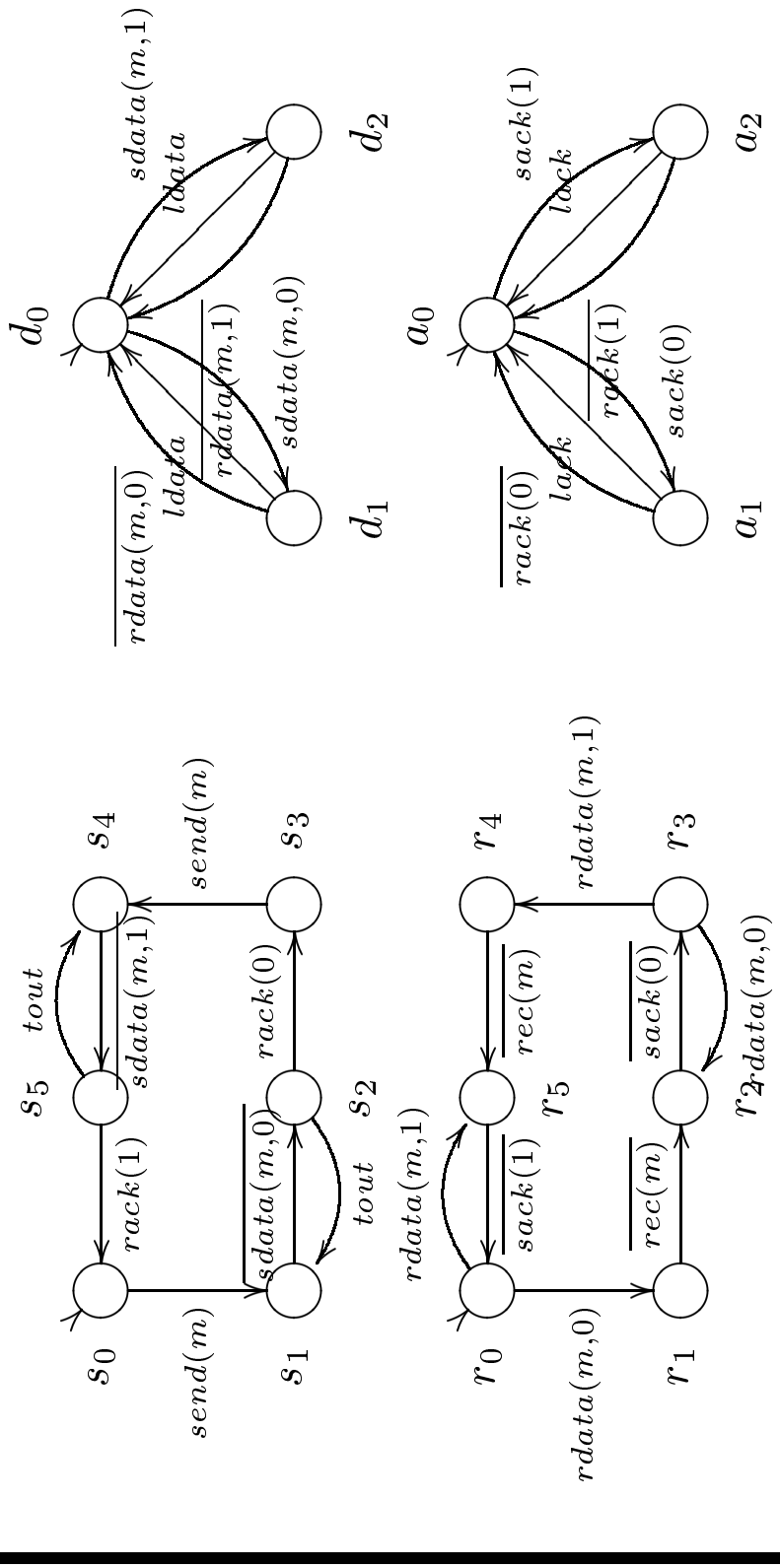
©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997–2000

CCS AND LABELLED TRANSITION SYSTEMS

- There is a clear connection between CCS and LTSs:
  - Each automaton in an LTS defines an agent. By defining which events in the figure are input events and which are output events, an agent is created which describes the LTS as a parallel composition of agents. Additionally, a restriction over input and output events has to be made.
  - Correspondinglym each CCS agent defines an automaton.
- This connection is going to be studied in the following.

©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997–2000

CCS AND LABELLED TRANSITION SYSTEMS



©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997–2000

CCS AND LABELLED TRANSITION SYSTEMS

- An agent expression corresponding the sender is as follows:

$$\begin{aligned} Sender &\stackrel{\text{def}}{=} send(m).Sending_1 \\ Sending_1 &\stackrel{\text{def}}{=} \overline{sdata(m,0)}.(tout.Sending_1 + rack(0).Sending_2) \\ Sending_2 &\stackrel{\text{def}}{=} send(m).Sending_3 \\ Sending_3 &\stackrel{\text{def}}{=} \overline{sdata(m,1)}.(tout.Sending_3 + rack(1).Sender) \end{aligned}$$

- The agents $Receiver$, $Data$, and Ack can be easily constructed (tutorial).
- Hence, an agent expression corresponding the AB protocol is given below:

$$AB \stackrel{\text{def}}{=} (Sender|Data|Receiver|Ack)\backslash\{sdata, rdata, sack, rack\}$$

©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997–2000

Operational semantics

- In the following we consider what is meant by the behaviour of an agent.
- It is useful to perceive a process in certain *states* and that state change is the effect of some *events* called *state transitions*.
- Let us consider the following agent:
  $Change \stackrel{\text{def}}{=} fifty.\overline{out}(20).\overline{out}(20).\overline{out}(10).Change.$
- The definition can also be written as given below:
  $Change \stackrel{\text{def}}{=} fifty.Change'$
  $Change' \stackrel{\text{def}}{=} .\overline{out}(20).Change''$
  $Change'' \stackrel{\text{def}}{=} \overline{out}(20).Change'''$
  $Change''' \stackrel{\text{def}}{=} \overline{out}(10).Change$
- The agents $Change, Change', Change'', Change'''$ can be thought of as representing the states of the system. The the functions $fifty$, $\overline{out}(20)$, $\overline{out}(10)$ are state trasitions.
- From now on we will trat the agent, state, and the state of the agent as synonyms.
- State transitions are depicted as given below:

$$P \stackrel{a}{\rightarrow} Q$$

©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997–2000

Communication

- What is the formal definition of a communication event?
- Let us assume that the following agent:
  $Person \stackrel{\text{def}}{=} \overline{fifty}.Person'$
  $Person' \stackrel{\text{def}}{=} \ldots$
  describes a person who wants to change 50 pence to smaller coins.
- The agent can do a state transaction: $Person \stackrel{\overline{fifty}}{\rightarrow} Person'$
- Correspondingly: $Change \stackrel{fifty}{\rightarrow} Change'$
- In other words, it is possible to do the functions $fifty$ and $\overline{fifty}$ in parallel or the processes are able to communicate.
- $Change|Person \stackrel{?}{\rightarrow} Change'|Person'$.
- What kind of an event os ? ?
- CCS defines the communication event as an invisible event. The environment of the agents taking part in the communication event is unable to observe said event. An invisible event is denoted τ.

©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997–2000

Rules

- Nil: Nil there is no rule for the agent.
- Prefixing: $a.P$ is an agent which can do a first and the P.
  - $a.P \xrightarrow{a} P$ (Act)
  - $l(x).P \xrightarrow{l_v} P\{v/x\}$ (Act')
  - $\overline{l}(x).P \xrightarrow{l_v} P$ (Act")
- Sum: Agent $P + Q$ can do what P and Q can do.
  - $P \xrightarrow{a} P' \Longrightarrow P + Q \xrightarrow{a} P'$ (Sum1)
  - $Q \xrightarrow{a} Q' \Longrightarrow P + Q \xrightarrow{a} Q'$ (Sum2)
- Composition: Agent $P|Q$ functions so that the agents P and Q can function independently or by communicating with each other:
  - $P \xrightarrow{a} P' \Longrightarrow P|Q \xrightarrow{a} P'|Q$ (Com1)
  - $Q \xrightarrow{a} Q' \Longrightarrow P|Q \xrightarrow{a} P|Q'$ (Com2)
  - $P \xrightarrow{l} P'$ ja $Q \xrightarrow{\overline{l}} Q' \Longrightarrow P|Q \xrightarrow{\tau} P'|Q'$ (Com3)
- Restriction: Agent $P\backslash L$ functions as agent P, disclosing functions $l, \overline{l} \in L$:
  - If $a, \overline{a} \notin L$ then $P \xrightarrow{a} P' \Rightarrow P\backslash L \xrightarrow{a} P'\backslash L$ (Res)

©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997–2000

Rules

- Composition: Agent $P|Q$ functions so that agents P and Q can function independently or communicate:
  - $P \xrightarrow{a} P' \Longrightarrow P|Q \xrightarrow{a} P'|Q$ (Com1)
  - $Q \xrightarrow{a} Q' \Longrightarrow P|Q \xrightarrow{a} P|Q'$ (Com2)
  - $P \xrightarrow{l} P'$ ja $Q \xrightarrow{\bar{l}} Q' \Longrightarrow P|Q \xrightarrow{\tau} P'|Q$ (Com3)
- Rules: Agent $P\backslash L$ functions as agent P, disclosing functions $l, \bar{l} \in L$:
  - If $a, \bar{a} \notin L$ then $P \xrightarrow{a} P' \Rightarrow P\backslash L \xrightarrow{a} P'\backslash L$ (Res)
- Renaming: Agent $P[f]$ is an agent the functioning of which is the same as that of P but all names have been renamed with the help of f:
  - $P \xrightarrow{a} P' \Rightarrow P[f] \xrightarrow{f(a)} P'[f]$ (Rel)

  An additional requirement is $f(\tau) = \tau$.

©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997–2000

Rules

- Choice: Agent $\text{if } e \text{ then } P \text{ else } Q$ functions as P or Q depending on the value of expression e.
  - If e is true then:
    $P \xrightarrow{a} P' \Rightarrow \text{if } e \text{ then } P \text{ else } Q \xrightarrow{a} P'$ (Cond1)
  - If e is false then:
    $Q \xrightarrow{a} Q' \Rightarrow \text{if } e \text{ then } P \text{ else } Q \xrightarrow{a} Q'$ (Cond2)
- Definition expression: $P \stackrel{\text{def}}{=} A$ means that the behaviour of P is defined with A:
  - $A \xrightarrow{a} P' \Rightarrow P \xrightarrow{a} P'$ (Rec)

©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997–2000

Semantics

- Rule based deduction is often denoted as given below:

$$\frac{P_1 \stackrel{a_1}{\rightarrow} P_1' \ldots P_n \stackrel{a_n}{\rightarrow} P_n'}{P \stackrel{a}{\rightarrow} P'} \text{(X)}$$

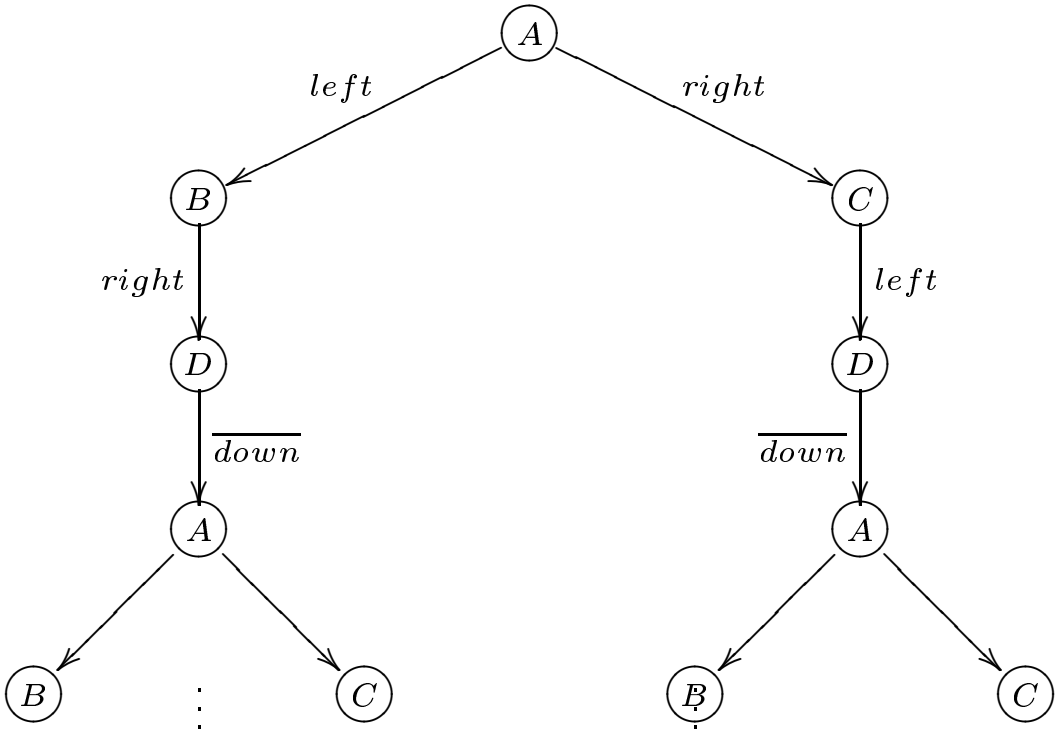
- The expression $Change|Person \stackrel{\tau}{\rightarrow} Change'|Person'$ can now be derived in the following manner:

$$\frac{\dfrac{}{Person \stackrel{\overline{fifty}}{\rightarrow} Person'} \text{(Act)} \quad \dfrac{}{Change \stackrel{fifty}{\rightarrow} Change'} \text{(Act)}}{Change|Person \stackrel{\tau}{\rightarrow} Change'|Person'} \text{(Com3)}$$

©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997–2000

Semantics

- Often, transition trees and transition diagrams can be used instead of derivations.
- The agent A
  $A \stackrel{\text{def}}{=} left.right.\overline{down}.A + right.left.\overline{down}.A$
  has a transition tree given below:



©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997–2000

Semantics

- A corresponding transition diagram is as follows:

©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997–2000

Equivalent behaviour

- What do we mean when we say two systems P and Q have the same behaviour?
- We pursue to create a relation between the states and events of systems.
- *Observational equivalence*: An equivalence relation between communication events of two systems.
- *Observational congruence*: Observational equivalence which is compatible with the operations of a process algebra.
- A relation should not be too strong or too weak.
- An observational congruence should meet the following criteria:
  - $P = Q$ if Q is a refinement of P.
  - If $P = Q$ and P' is a subsystem of P for which $P' = R$ is valid, then $P[R/P'] = Q$

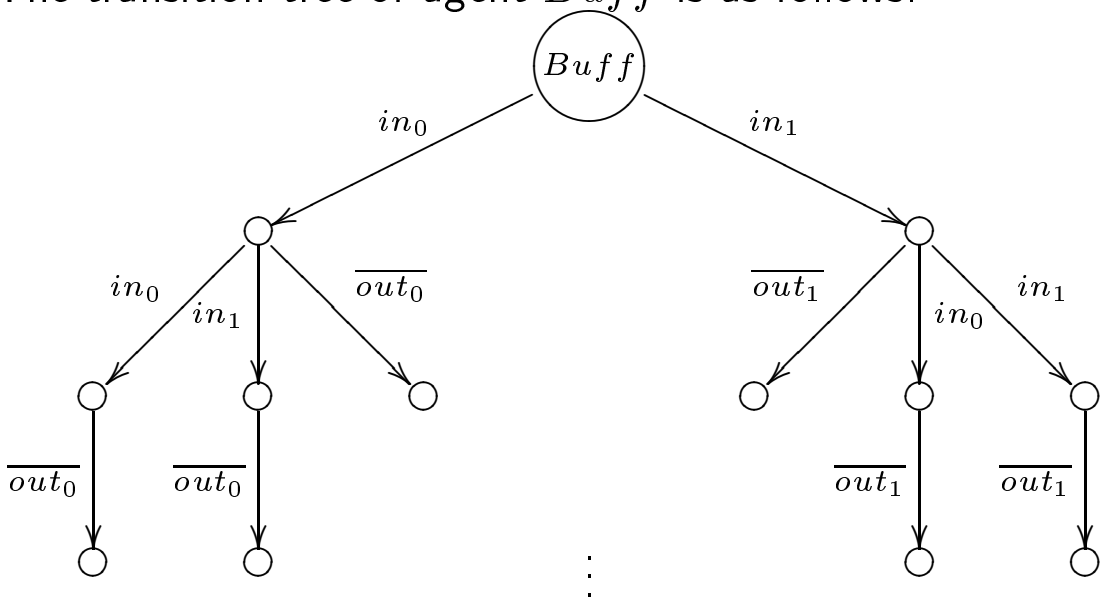
©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997–2000

Equivalent behaviour

Consider the agents given below:

$$
\begin{aligned}
Buff &\stackrel{\text{def}}{=} in(x).Buff'(x) \\
Buff'(x) &\stackrel{\text{def}}{=} in(y).Buff''(x,y) + \overline{out(x)}.Buff \\
Buff''(x,y) &\stackrel{\text{def}}{=} \overline{out}(x).Buff'(y)
\end{aligned}
$$

The transition tree of agent $Buff$ is as follows:



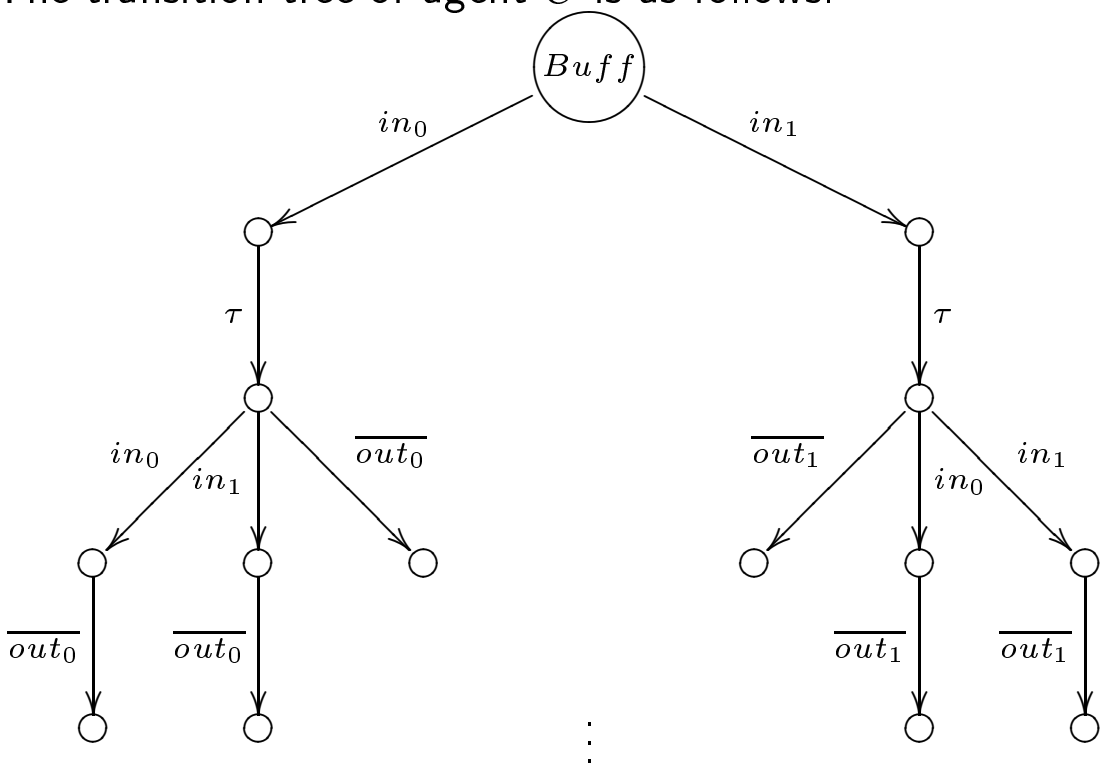
©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997–2000

Equivalent behaviour

Consider the agents given below:

$$
\begin{aligned}
Cell &\stackrel{\text{def}}{=} in(x).Cell'(x) \\
Cell'(x) &\stackrel{\text{def}}{=} \overline{out}(x).Cell \\
C1 &\stackrel{\text{def}}{=} Cell[\overline{mid}/\overline{out}] \\
C2 &\stackrel{\text{def}}{=} Cell[mid/in] \\
C &\stackrel{\text{def}}{=} (C1 \mid C2)\backslash\{mid\}
\end{aligned}
$$

The transition tree of agent C is as follows:



©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997–2000

Equivalent behaviour

- Since τ-events are invisible it can be said that agents C and $Buff$ have the same observable behaviour.
- A consequence is that agent C can be replaced with agent $Buff$ in any system and vice versa.
- Replacability can be formalized with the relation $=$.
- The relation should meet with the following criteria:

(1) $P = P$,

(2) If $P = Q$ then $Q = P$,

(3) If $P = Q$ and $Q = R$ then $P = R$,

(4) If $P = Q$ then

$$a.P = a.Q$$

$$P + R = Q + R$$

$$P|R = Q|R$$

$$P\backslash L = Q\backslash L$$

$$\text{if } e \text{ then } P \text{ else } R = \text{ if } e \text{ then } Q \text{ else } R$$

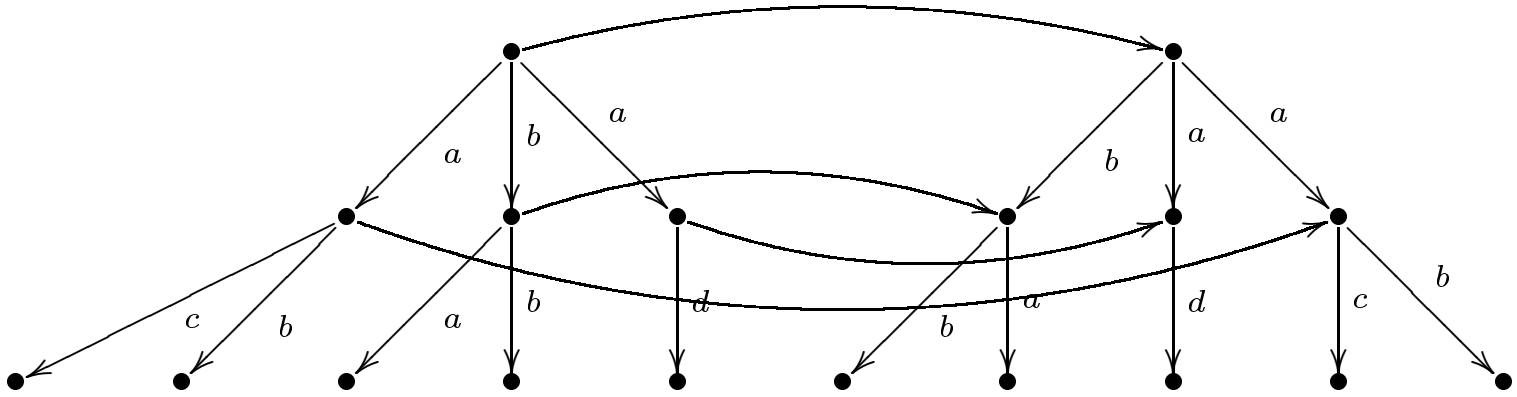
if $A \stackrel{\text{def}}{=} P$ and $B \stackrel{\text{def}}{=} Q$ then $A = B$.

- A relation meeting with criteria (1)-(3) is an **OBSERVATIONAL EQUIVALENCE**.
- A relation meeting with criteria (1)-(4) is an **OBSERVATIONAL CONGRUENCE**

©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997–2000

Equivalent behaviour

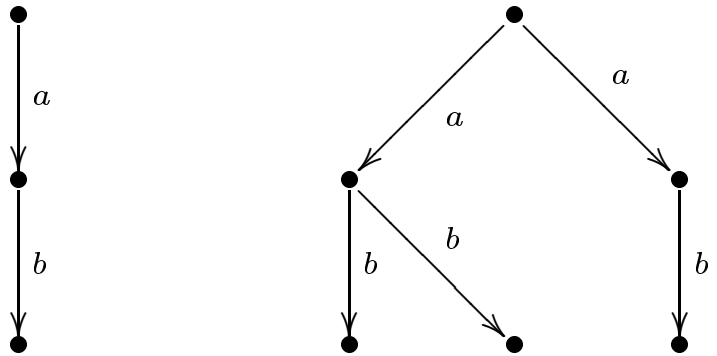
- What is a suitable relation for it?
  - Syntactic identity?
  - Too strong: C and $Buff$ would not be congruent.
  - Isomorphism of transition trees?



©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997–2000

Equivalent behaviour

- Too strong because the following trees:



  are not equivalent but we would want agents $a.b.Nil$ and $a.b.Nil + a.(b.Nil + b.Nil)$ to be equivalent.
- Agents are equivalent if they are able to do the same sequence of state transitions?

$$P = Q \text{ if } P \xrightarrow{s} P' \text{ iff } Q \xrightarrow{s} Q'$$

  Now the above trees are equivalent.

©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997–2000

Equivalent behaviour

- However, the equivalence is too weak:

$$a.(b.Nil + c.Nil) = a.b.Nil + a.c.Nil$$

  the right hand agent is able to do a so that c is not a possible successor event.
- An equivalence should preserve the "time point" of choice.
- This means:
  - Equivalent agents have to be able to do the same functions.
  - After a certain function the subsequent states should be 'equivalent.'

©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997–2000

Equivalent behaviour

- Definition (*): P and Q are equivalent iff for each event a the following holds:
  - If P' such that $P \xrightarrow{a} P'$ then there exists Q' such that $Q \xrightarrow{a} Q'$ and P' and Q' are equivalent, or
  - If Q' such that $Q \xrightarrow{a} Q'$ then there exists P' such that $P \xrightarrow{a} P'$ and Q' and P' are equivalent.
- This is a bad definition: An equivalence is defined as an equivalence.
- However, there prove to exist relations for which the definition (*) holds. Amongst these relations there is the 'largest' one which also is a congruence.
- On the other hand, the definition (*) ignores τ events.
- E.g., agents $\tau.Nil$ and Nil as well as agents $a.Nil$ and $a.\tau.Nil$ are not equivalent.

©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997–2000

Equivalent behaviour

- Notation: $P \stackrel{a}{\Rightarrow} Q$ iff $P \stackrel{<\tau>^m <a> <\tau>^n}{\rightarrow} Q$
- Example: If $P \stackrel{\text{def}}{=} \tau.a.\tau.Nil$ then $P \stackrel{a}{\Rightarrow} Nil$
- N.B.: $P \stackrel{\tau}{\Rightarrow} Q$ iff P is able to change to Q by executing *at least* one τ event.
- Definition (**): P and Q are equivalent iff for each event a the following holds:
  - If P' such that $P \stackrel{a}{\rightarrow} P'$ then there exists Q' such that $Q \stackrel{a}{\Rightarrow} Q'$ and P' and Q' are equivalent, or
  - If Q' such that $Q \stackrel{a}{\rightarrow} Q'$ then there exists P' such that $P \stackrel{a}{\Rightarrow} P'$ and Q' and P' are equivalent.
- This not what we wanted because, e.g., agents $\tau.Nil$ and Nil as well as $a.Nil$ and $a.\tau.Nil$ are still not equivalent. (Why?)

©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997–2000

Equivalent behaviour

- Definition (***): P and Q are equivalent if for each event a the following holds:
  - If P' such that $P \xrightarrow{a} P'$ then *either* (1) there exists Q' such that $Q \stackrel{a}{\Rightarrow} Q'$ and P' and Q' are equivalent, *or* (2) $a = \tau$ and P' and Q are equivalent, and correspondingly
  - If Q' such that $Q \xrightarrow{a} Q'$ then *either* (1) there exists P' such that $P \stackrel{a}{\Rightarrow} P'$ and Q' and P' are equivalent, *or* (2) $a = \tau$ and Q' and P are equivalent.
- This definition agrees with our intuition about the equivalence of two agents.

©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997–2000

Equivalent behaviour

- Definition (***) is not directly applicable as the definition of observation equivalence.
- It proves that there is a *largest* relation for which definition (***) holds. It will be denoted $\approx$..
- Definition: Relation $\approx$ is an **OBSERVATIONAL EQUIVALENCE** when $P \approx Q$ if for each event a the following holds:
  - If P' such that $P \xrightarrow{a} P'$ then *either* (1) there exists Q' such that $Q \stackrel{a}{\Rightarrow} Q'$ and $P' \approx Q'$, *or* (2) $a = \tau$ and $P' \approx Q$, and correspondingly
  - If Q' such that $Q \xrightarrow{a} Q'$ then *either* there exists P' such that $P \stackrel{a}{\Rightarrow} P'$ and $Q' \approx P'$, *or* (2) $a = \tau$ and $Q' \approx P$.

  Moreover, $\approx$ is the largest relation of its kind, which means that if $\mathcal{R}$ there is another relation fulfilling definition (***) and if $P\mathcal{R}Q$ then $P \approx Q$.

©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997–2000

Equivalent behaviour

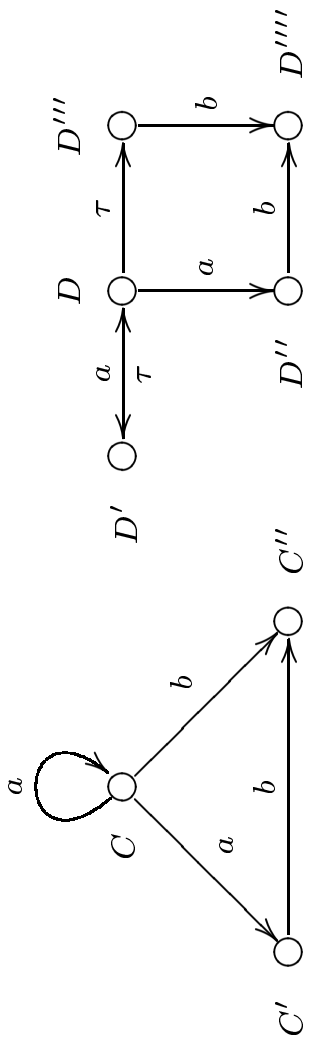
- $\approx$ is 'almost' an observational congruence.
- E.g., $P \approx Q \Rightarrow P + R \approx Q + R$ never holds because $\tau.a.Nil \approx a.Nil$ but $\tau.a.Nil + l.Nil \not\approx a.Nil + l.Nil$
- Observational congruence is the largest relation contained in observational equivalence and it is a congruence of agents.
- Let us denote it $P = Q$.

©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997–2000

EXAMPLE

$$C \stackrel{\text{def}}{=} a.C + a.b.Nil + b.Nil$$

$$D \stackrel{\text{def}}{=} a.\tau.D + \tau.b.Nil + a.b.Nil$$



$C \cong D$?

©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997–2000

Properties of observational equivalence

- We will define laws with which a two agent expression can be simplified.
- There are four groups:
  - *Dynamic laws* define properties of dynamic operators:
    - Prefixing and
    - Sum.
  - *Static laws* define properties of static operators:
    - Parallel composition,
    - Restriction, and
    - Renaming.
  - *Expansion principle* with which the connection between the dynamic and static laws is defined. ja staattisten lakien yhteys.
  - *Recursion rules* defining the properties of recursion.

©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997–2000

Dynamic laws

1.

$$P + Q = Q + P \quad (1)$$

$$P + (Q + R) = (P + Q) + R \quad (2)$$

$$P + P = P \quad (3)$$

$$P + Nil = P \quad (4)$$

2.

$$a.\tau.P = a.P \quad (5)$$

$$P + \tau.P = \tau.P \quad (6)$$

$$a.(P + \tau.Q) + a.Q = a.(P + \tau.Q) \quad (7)$$

$$P + \tau.(P + Q) = \tau.(P + Q) \quad (8)$$

©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997–2000

Non-laws

- The following laws do *NOT* hold even if they feel intuitively right:
  - $$P \stackrel{?}{=} \tau.P$$
  - 'distributivity'
    $$a.(P + Q) \stackrel{?}{=} a.P + a.Q$$

©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997–2000

Static laws

1.

$$P|Q = Q|P \quad (9)$$

$$P|(Q|R) = (P|Q)|R \quad (10)$$

$$P|Nil = P \quad (11)$$

2.

$$P\backslash L = P \text{ iff } sort(P) \cap (L \cup \overline{L}) = \emptyset \quad (12)$$

$$((P\backslash L)\backslash K) = P\backslash(L \cup K) \quad (13)$$

$$(P[f])\backslash L = (P\backslash f^{-1}(L))[f] \quad (14)$$

$$(P|Q)\backslash L = (P)\backslash L)|(Q)\backslash L) \quad (15)$$

$$\text{iff } sort(P) \cap \overline{sort(Q)} \cap (L \cup \overline{L}) = \emptyset \quad (16)$$

©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997–2000

Static laws

1.

$$P[Id_L] = P \tag{17}$$

$$P[f] = P[g] \text{ if } f(sort(P)) = g(sort(P)) \tag{18}$$

$$(P[f])[g] = P[g \circ f] \tag{19}$$

$$(P|Q)[f] = (P[f])|(Q[f]) \tag{20}$$

$$\text{if } f \text{ is injective for the set } sort(P|Q) \cup \overline{sort(P|Q)} \tag{21}$$

2.

$$P[k/l] = P \text{ if } l, \overline{l} \notin sort(P) \tag{22}$$

$$P\backslash l = (P[k/l])\backslash k \text{ if } k, \overline{k} \notin sort(P) \tag{23}$$

$$(P\backslash L)[k/l] = (P[k/l])\backslash L \text{ if } k, l \notin L \cup \overline{L} \tag{24}$$

$$\tag{25}$$

©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997–2000

Expansion principle

- Often the easiest approach is to desribe the system as a parallel composition of its parts and restriction: $(P_1|P_2|\ldots|P_n)\backslash L$
- Expansion principle expresses how static operator can be opened up to dynamic operators.
- Dynamic laws help us derive an equivalent agent.
- Expansion prociple gives process algebra an **interleaving semantics**:

  PA "concurrency = choice + sequential composition"

  PV "concurrency = causal independence" ("true" concurrency)

©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997–2000

Expansion prociple

$$
\begin{aligned}
P_1|\ldots|P_n \quad = \quad & \Sigma\{a.(P_1|\ldots|P_i'|\ldots|P_n) \mid 1 \le i \le n,\ P_i \xrightarrow{a} P_i'\ \} \\
& +\Sigma\{\tau.(P_1|\ldots|P_i'|\ldots|P_j'|\ldots|P_n) \\
& |1 \le i \le j \le n, \exists l:\ P_i \xrightarrow{l} P_i'\ ,\ P_j \xrightarrow{\bar{l}} P_j'\ \} \qquad (26)
\end{aligned}
$$

$$
\begin{aligned}
(P_1[f_1]|\ldots|P_n[f_n])\backslash L \quad = \quad & \Sigma\{f_i(a).(P_1[f_1]|\ldots|P_i'[f_i]|\ldots|P_n[f_n])\backslash L \\
& |1 \le i \le n,\ P_i \xrightarrow{a} P_i'\ , f_i(a) \notin L \cup \overline{L}\} \\
& +\Sigma\{\tau.(P_1[f_1]|\ldots|P_i'[f_i]|\ldots|P_j'[f_j]|\ldots|P_n[f_n])\backslash L \\
& |1 \le i < j \le n, \exists l, k:\ P_i \xrightarrow{l} P_i'\ ,\ P_j \xrightarrow{k} P_j'\ \}, \\
& \overline{f_i(l)} = f_j(k)\} \qquad (27)
\end{aligned}
$$

©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997–2000

Recursion laws

- If $A \stackrel{\text{def}}{=} P$ then $A = P$.
- How to make it easier to analyse recursively defined agents?
- That is, how to prove that the following agents:

$$\begin{aligned} A &\stackrel{\text{def}}{=} l.A \\ B &\stackrel{\text{def}}{=} l.l.B \end{aligned}$$

  have the same behaviour.
- We will adopt the use of an agent variable.
- Let us assume that given is the equation $X = E$, where X is the agent variable and E does not contain other agent variables than X.
- Our goal is to prove that:
  1. Whenever P and Q are solutions to the equation, then $P = Q$, and
  2. A and B are solutions to the equation $X = E$.

©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997–2000

Recursion laws

- Item 2. can be verified with the above laws.
- Item 1. requires that certain conditions hold for the equation:
  - The agent variable of the equation $X = E$ is preceded by a visible event (X is guarded). E.g.,

    $$l.X + Y$$

    but not

    $$l.X + X.$$

  - So as to have unambiguous solutions, sequentiality is required. X is sequential in E iff no instatiation of X is located 'within' a static operator. E.g.,

    $$X + a.Nil, a.X + (Nil|b.Y)$$

    but not

    $$a.X|b.Nil, a.X + (a.X|Y)\backslash b$$

©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997–2000

Recursion laws

- Law 9B
  Let I be a set of indeces, $\{X_i|i \in I\}$ a set of agent variables and $\{E_i|i \in I\}$ a set of agent expressions such that $Var(\{E_i|i \in I\}) = \{X_i|i \in I\}$. Moreover, we assume that for each variable X_i there is a preceding event and each variable X_i is sequential in the expression E_j. This means that if $\{P_i|i \in I\}$ and $\{Q_i|i \in I\}$ are agent groups such that for each i both P_i and Q_i are solutions to the equation $X_i = E_i$ then for all $i \in I$ $P_i = Q_i$.

©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997–2000

Recursion laws

- Consider the following agents:

$$\begin{aligned} Buff &\stackrel{\text{def}}{=} in_0.B_0 + in_1.B_1 \\ B_i &\stackrel{\text{def}}{=} in_0.B_{i0} + in_1.B_{i1} + \overline{out_i}.Buff \\ B_ij &\stackrel{\text{def}}{=} \overline{out_i}.Buff_j \end{aligned}$$

and

$$\begin{aligned} A &\stackrel{\text{def}}{=} in_0.A_0 + in_1.A_1 \\ A_i &\stackrel{\text{def}}{=} \overline{mid_i}.A \\ A' &\stackrel{\text{def}}{=} mid_0.A'_0 + mid_1.A'_1 \\ A'_i &\stackrel{\text{def}}{=} \overline{out_i}.A \\ C &\stackrel{\text{def}}{=} (A \mid A')\backslash\{mid_0, mid_1\} \end{aligned}$$

©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997–2000

Recursion laws

- To prove that $C = Buff$ we have to show that both C and $Buff$ are solutions to the system of equations given below:

$$
\begin{aligned}
X_1 &= in_0.X_2 + in_1.X_3 \\
X_2 &= in_0.X_4 + in_1.X_5 + \overline{out_0}.X_1 \\
X_3 &= in_0.X_6 + in_1.X_7 + \overline{out_1}.X_1 \\
X_4 &= \overline{out_0}.X_2 \\
X_5 &= \overline{out_0}.X_3 \\
X_6 &= \overline{out_1}.X_2 \\
X_7 &= \overline{out_1}.X_3
\end{aligned}
$$

- By assigment $Buff, B_0, \ldots$ we can see that $Buff$ is a solution of the system of equations.
- Correspondingly, by applying the expansion pronciple iteratively the agent C is chaged into a form from which we can deduce that it is a solution to the system of equations as well.

©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997–2000

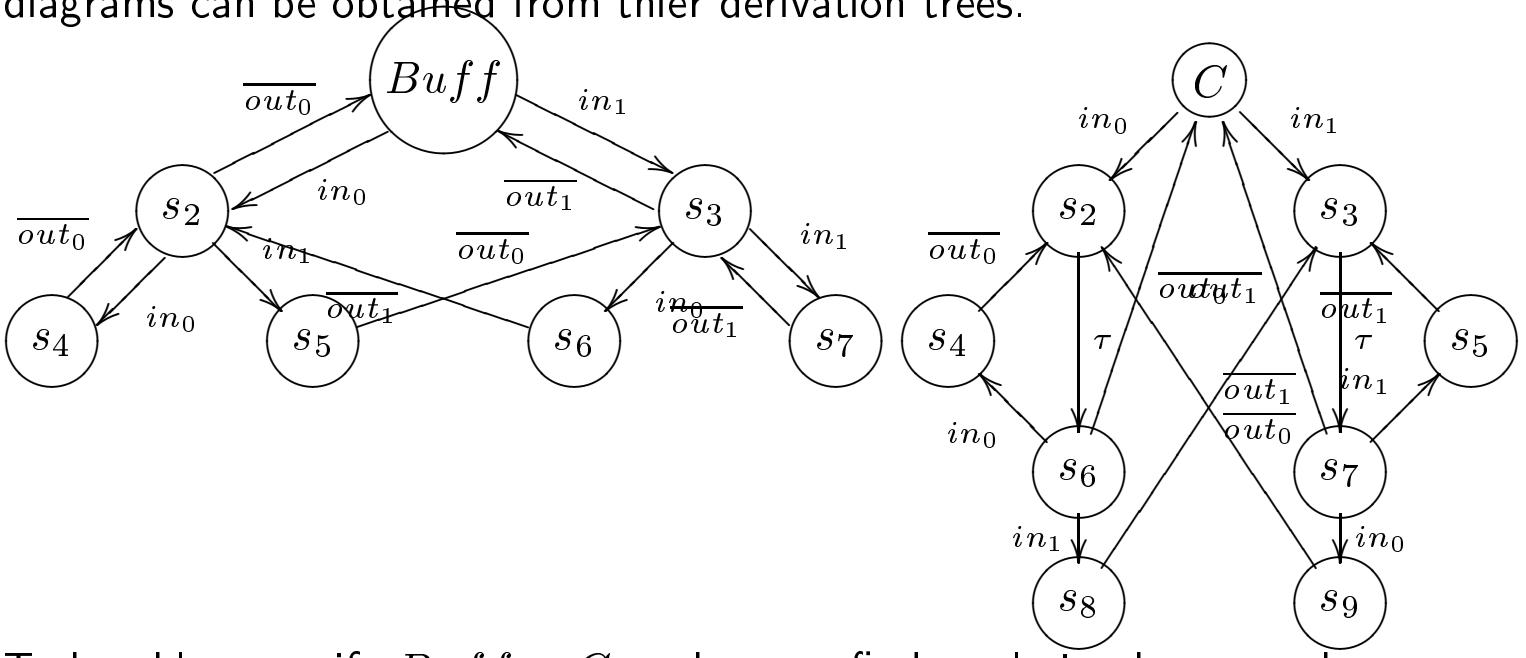
Bisimulation

- Bisimulation is method to check if two agents are observation equivalent.
- Together with the laws of observation congruence we now have a mthod to analyse the behaviour of agents.
- (***) Observation equivalence $\approx$ is the largest relation such that $P \approx Q$ iff for each event a the following holds:
  - If P' such that $P \xrightarrow{a} P'$ then *either*
    1. There exists Q' such that $Q \stackrel{a}{\Rightarrow} Q'$ and $P' \approx Q'$, *or*
    2. $a = \tau$ and $P' \approx Q$, and correspondingly
  - If Q' such that $Q \xrightarrow{a} Q'$ then *either*
    1. There exists P' such that $P \stackrel{a}{\Rightarrow} P'$ and $Q' \approx P'$, *or*
    2. $a = \tau$ and $Q' \approx P$.

©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997–2000

Bisimulation

- Yet again, let us consider the agents $Buff$ and C. The following transition diagrams can be obtained from thier derivation trees:



- To be able to verify $Buff \approx C$ we have to find a relation between the states of the system

©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997–2000

Bisimulation

- When we start by assuming that $(Buff, C)$ is such a pair, by applying the definition (***) the following matches can be found:

$$\begin{aligned} R \;=\; & \{(Buff, C), \\ & (B_0, (A_0|A')\backslash L), (B_0, (A|A'_0)\backslash L), \\ & (B_1, (A_1|A')\backslash L), (B_1, (A|A'_1)\backslash L), \\ & (B_{00}, (A_0|A'_0)\backslash L), (B_{01}, (A_1|A'_0)\backslash L), \\ & (B_{10}, (A_1|A'_0)\backslash L), (B_{10}, (A_0|A'_1)\backslash L), \\ & (B_{11}, (A_1|A'_1)\backslash L)\} \end{aligned}$$

- This match is part of all of observation equivalence.

©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997–2000

Bisimulation

- The following holds:
  If $(P, Q) \in R$ then for all events a:
  - If P' such that $P \xrightarrow{a} P'$ then *either*
    1. There exists Q' such that $Q \stackrel{a}{\Rightarrow} Q'$ and $(P', Q') \in R$, *or*
    2. $a = \tau$ and $(P', Q) \in R$, and correspondingly
  - If Q' such that $Q \xrightarrow{a} Q'$ then *either*
    1. There exists P' such that $P \stackrel{a}{\Rightarrow} P'$ and $(Q', P') \in R$, *or*
    2. $a = \tau$ ja $(Q', P) \in R$.
- This kind of a relation is a *BISIMULATION*.
- Bisimulation meets the criterion (***) from left to right only.
- However, there is simple method to upgrade it to observation equivalence.

©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997–2000

Bisimulation

- When does the following hold: $P \approx Q \Rightarrow P = Q$?
- The reason for the fact that not always $P \approx Q \Rightarrow P = Q$ is the outruling effect of the τ event.
- If P can be started with an invisible event it may prevent other possible behaviours.
- However, neither $Buff$ nor C are able to start with a τ evetn.
- This kind of an agent is *STABLE*.
- Stability principle:
  If P and Q are stable and $P \approx Q$ then $P = Q$.

©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997–2000

Bisimulation

- Stability principle is descriptive of the difference between observation equivalence and observation congruence.
- Observaton congruence can also be defined explicitly:
  $P = Q$ iff for all events a the following holds:
  - If P' such that $P \stackrel{a}{\rightarrow} P'$ then there exists Q' such that $Q \stackrel{a}{\Rightarrow} Q'$ and $P' \approx Q'$, and correspondingly
  - If Q' such that $Q \stackrel{a}{\rightarrow} Q'$ then there exists P' such that $P \stackrel{a}{\Rightarrow} P'$ and $Q' \approx P'$.

©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997–2000

Bisimulation

- Consider the following agents:

$$\begin{aligned} Sem0 &\stackrel{\text{def}}{=} get.Sem1 \\ Sem1 &\stackrel{\text{def}}{=} get.Sem2 + put.Sem0 \\ Sem2 &\stackrel{\text{def}}{=} get.Sem3 + put.Sem1 \\ Sem3 &\stackrel{\text{def}}{=} put.Sem2 \end{aligned}$$

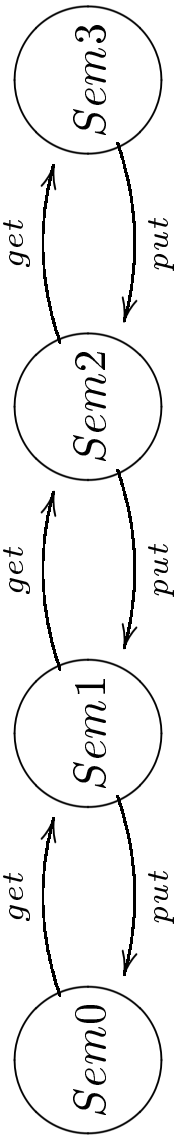
sekä

$$\begin{aligned} S &\stackrel{\text{def}}{=} get.S' \\ S' &\stackrel{\text{def}}{=} put.S \\ T &\stackrel{\text{def}}{=} S|S|S \end{aligned}$$

©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997–2000

BISIMULATION

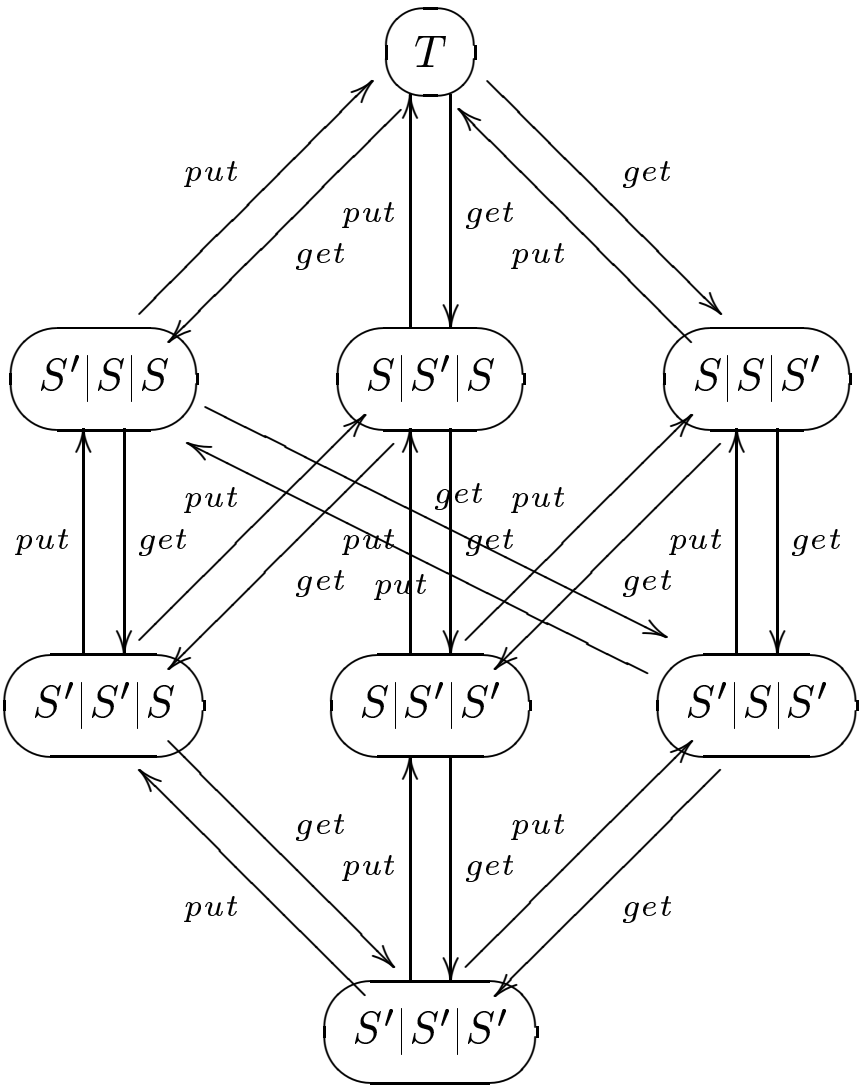
- $Sem0$ describes a ternary semaphore. Its transition diagram is given below:



©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997–2000

Bisimulation

- Correspondingly, the transition diagram of T is given below:



©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997–2000

Bisimulation

- We will prove that $Sem0 \approx T$.
- We are looking for a bisimulation containing the pair $(Sem0, T)$.
- Such a pair exists:

$$\begin{aligned} R \quad = \quad & \{(Sem0, T), \\ & (Sem1, S'|S|S), (Sem1, S|S'|S), \\ & (Sem1, S|S|S'), (Sem2, S'|S'|S), \\ & (Sem2, S'|S|S'), (Sem2, S|S'|S'), \\ & (Sem3, S'|S'|S')\} \end{aligned}$$

- However, it can be proved that

$$S'|S|S \approx S|S'|S \approx S|S|S'$$

and

$$S'|S'|S \approx S'|S|S' \approx S|S'|S'$$

- Was our effort wasted?

©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997–2000

Bisimulation

- The definiton of bisimulation can be modified so that while creating the bisimulation we can take advantage of the laws of observation congruence:
- If $(P, Q) \in R$ then for all events a the following holds:
  - If P' such that $P \stackrel{a}{\rightarrow} P'$ then *either*
    1. There exists P'', Q', Q'' such that $Q \stackrel{a}{\Rightarrow} Q'$ and $P' \approx P'', Q' \approx Q''$ and $(P'', Q'') \in R$, *or*
    2. $a = \tau$ and $(P', Q) \in R$, and correspondingly
  - If Q' such that $Q \stackrel{a}{\rightarrow} Q'$ then *either*
    1. there exists P' such that $P \stackrel{a}{\Rightarrow} P'$ and $Q' \approx Q'', P' \approx P''$ $(P'', Q'') \in R$, *or*
    2. $a = \tau$ and $(Q', P) \in R$.

©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997–2000

Example: Alternating bit protocol

- Yet again, let us consider the alternating bit protocol.
- To prove the protocol:
  - The protocol is meant work as a buffer.
  - This means:
    1. Construct an agent AB for the protocol.
    2. Construct an agent $Buff$ for the buffer.
    3. Look for a bisimulation to prove that $AB \approx Buff$.

©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997–2000

Example: Alternating bit protocol

- Messages are sent tagged with bits 0 and 1.
- Sender process: Once started, a message is sent with bit b.
  - If a message is lost the process time-outs and the message is resent, tagged with bit b.
  - If the process receives bit b as an acknowledgement, it will send a new message, tagged with bit $\hat{b} = 1 - b$.
  - If the process receives bit $\hat{b}$ as an acknowledgement it will be forgotten.
- Receiver process: The process receives a message tagged with bit b. The bit b sent back as an ackowledgement.
  - If the ackowledgement is lost the process time-outs and the acknowledgement is resent.
  - If the process receives a new message tagged with bit $\hat{b}$ this will be ackowledged.
  - If the process receives the previous message again, tagged with bit b, it will be forgotten.
- N.B.: Here also the reveiver process is able to time-out. (Cf. the ABP model of the first lecture.)

©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997–2000

Example: Alternating bit protocol

- The expressions for the agents are given below:

$$
\begin{aligned}
Send(b) &\stackrel{\text{def}}{=} \overline{\texttt{send}_b}.\overline{\texttt{time}}.Sending(b) \\
Sending(b) &\stackrel{\text{def}}{=} \texttt{timeout}.Send(b) + \texttt{ack}_b.\texttt{timeout}.Accept(\hat{b}) \\
&\quad + \texttt{ack}_{\hat{b}}.Sending(b) \\
Accept(b) &\stackrel{\text{def}}{=} \texttt{accept}.Send(b) \\
Reply(b) &\stackrel{\text{def}}{=} \overline{\texttt{reply}_b}.\overline{\texttt{time}}.Replying(b) \\
Replying(b) &\stackrel{\text{def}}{=} \texttt{timeout}.Reply(b) + \texttt{trans}_{\hat{b}}.\texttt{timeout}.Deliver(\hat{b}) \\
&\quad + \texttt{trans}_b.Replying(b) \\
Deliver(b) &\stackrel{\text{def}}{=} \overline{\texttt{deliver}}.Reply(b) \\
Timer &\stackrel{\text{def}}{=} \texttt{time}.\overline{\texttt{timeout}}.Timer
\end{aligned}
$$

©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997–2000

Example: Alternating bit protocol

- The channels are easier to describe with transition rules, given below:

$$Ack(bs) \overset{\overline{\mathtt{ack}_b}}{\to} Ack(s) \qquad Trans(sb) \overset{\overline{\mathtt{trans}_b}}{\to} Trans(s)$$

$$Ack(s) \overset{\mathtt{reply}_b}{\to} Ack(sb) \qquad Trans(s) \overset{\mathtt{send}_b}{\to} Trans(bs)$$

$$Ack(sbt) \overset{\tau}{\to} Ack(st) \qquad Trans(tbs) \overset{\tau}{\to} Trans(ts)$$

$$Ack(sbt) \overset{\tau}{\to} Ack(sbbt) \qquad Trans(tbs) \overset{\tau}{\to} Trans(tbbs)$$

- Now the following simplifications are worthwhile.

$$Send'(b) \overset{\text{def}}{=} (Send(b)|Timer)\backslash\{\mathtt{time}, \mathtt{timeout}\}$$

$$Reply'(b) \overset{\text{def}}{=} (Reply(b)|Timer)\backslash\{\mathtt{time}, \mathtt{timeout}\}$$

©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997–2000

Example: Alternating bit protocol

- The agent expression are now in the following form:

$$
\begin{aligned}
Send(b) &\stackrel{\text{def}}{=} \overline{\texttt{send}_b}.Sending(b) \\
Sending(b) &\stackrel{\text{def}}{=} \tau.Send(b) + \texttt{ack}_b.Accept(\hat{b}) + \texttt{ack}_{\hat{b}}.Sending(b) \\
Accept(b) &\stackrel{\text{def}}{=} \texttt{accept}.Send(b) \\
Reply(b) &\stackrel{\text{def}}{=} \overline{\texttt{reply}_b}.Replying(b) \\
Replying(b) &\stackrel{\text{def}}{=} \tau.Reply(b) + \texttt{trans}_{\hat{b}}.Deliver(\hat{b}) + \texttt{trans}_b.Replying(b) \\
Deliver(b) &\stackrel{\text{def}}{=} \overline{\texttt{deliver}}.Reply(b)
\end{aligned}
$$

- The protocol now has the following definition:

$$AB = Accept(\hat{b})|Trans(\epsilon)|Ack(\epsilon)|Reply(b)$$

©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997–2000

Example: Alternating bit protocol

- The protocol has to implement the following specification:

$$\begin{aligned} Buff &\stackrel{\text{def}}{=} \texttt{accept}.Buff' \\ Buff' &\stackrel{\text{def}}{=} \overline{\texttt{deliver}}.Buff \end{aligned}$$

- To prove this we have to find a bisimulation such that:

$$AB \approx Buff$$

- What is notable is that

$$AB \neq Buff$$

  because AB is not stable.

©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997–2000

Example: Alternating bit protocol

- The state space can be split up so that the following bisimulation candidate is obtained:

$$Accept(\hat{b})|Trans(b^n)|Ack(b^p)|\left\{\begin{array}{l} Reply(b) \\ Replying(b) \end{array}\right. \quad Buff$$

$$\left.\begin{array}{l} Send(\hat{b}) \\ Sending(\hat{b}) \end{array}\right\}|Trans(\hat{b}^m b^n)|Ack(b^p)|\left\{\begin{array}{l} Reply(b) \\ Replying(b) \end{array}\right. \quad Buff'$$

$$\left.\begin{array}{l} Send(\hat{b}) \\ Sending(\hat{b}) \end{array}\right\}|Trans(\hat{b}^m)|Ack(b^p)|Deliver(\hat{b}) \quad Buff'$$

$$\left.\begin{array}{l} Send(\hat{b}) \\ Sending(\hat{b}) \end{array}\right\}|Trans(\hat{b}^m)|Ack(b^p\hat{b}^q)|\left\{\begin{array}{l} Reply(\hat{b}) \\ Replying(\hat{b}) \end{array}\right. \quad Buff$$

©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997–2000

Example: Alternating bit protocol

- E.g., consider the following pair:

$$(Sending(\hat{b})|Trans(\hat{b}^m b^n)|Ack(b^p)|Reply(b), Buff')$$

- Its left element is able to:
  - Perform τ events corresponding the repeat of $Trans$ or Ack events. All matching successor states belong to the same group so we can pick the state $Buff'$ as its own τ successor.
  - The τ event

$$Sending(\hat{b}) \xrightarrow{\tau} Send(\hat{b})$$

    takes us to another group which is in relation with $Buff'$.
  - If $Sending(b)$ performs the event $\overline{\mathtt{ack}_b}$ then, correspondingly, $Ack(b^p)$ is able to perform $\mathtt{ack}_b$, and the created τ takes us to the same group.
  - Eventually, if $Reply(b)$ performs the event $\overline{\mathtt{reply}_b}$ then, correspondingly, $Ack(b^p)$ is able to perform the event $\mathtt{reply}_b$ whereupon the τ event takes us to a different group which is, however, in a relation with $Buff'$.

©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997–2000

Example: Alternating bit protocol

- On the other hand, $Buff'$ can only perform

$$Buff' \stackrel{\overline{\texttt{deliver}}}{\rightarrow} Buff.$$

Equally well, the protocol is able to perform $\overline{\texttt{reply}_b}$, $\texttt{trans}_b$ (n times), and $\texttt{trans}_{\hat{b}}$ producing τ events, and finally the event $\overline{\texttt{deliver}}$. The state we obtain is

$$Sending(\hat{b})|Trans(\hat{b}^{m-1})|Ack(b^{p+1})|Reply(\hat{b}),$$

which is in relation with the agent $Buff$.

©Johan Lilius, Keijo Heljanko, and Esa Kettunen 1997–2000