Communication and Concurrency: CCS

R. Milner, “A Calculus of Communicating Systems”, 1980

Why calculi?

- Prove properties on programs and languages
- Principle: tiny syntax, small semantics, to be handled on paper or mechanically
- Prove properties on the principles of a language or a programming paradigm

- Examples: lambda calculus, sigma calculus, …

Static semantics : examples

- Checks non-syntactic constraints
- compiler front-end :
  - declaration and utilisation of variables,
  - typing, scoping, … static typing => no execution errors ???
- or back-ends :
  - optimisers
- defines legal programs :
  - Java byte-code verifier

What can we do/know about a program without executing it?

Dynamic semantics

- Gives a meaning to the program (a semantic value)
- Describes the behaviour of a (legal) program
- Defines a language interpreter

|- e -> e ’

let i=3 in 2*i -> 6

Objective = to prove properties on
Program execution
(determinacy, subject reduction, …)

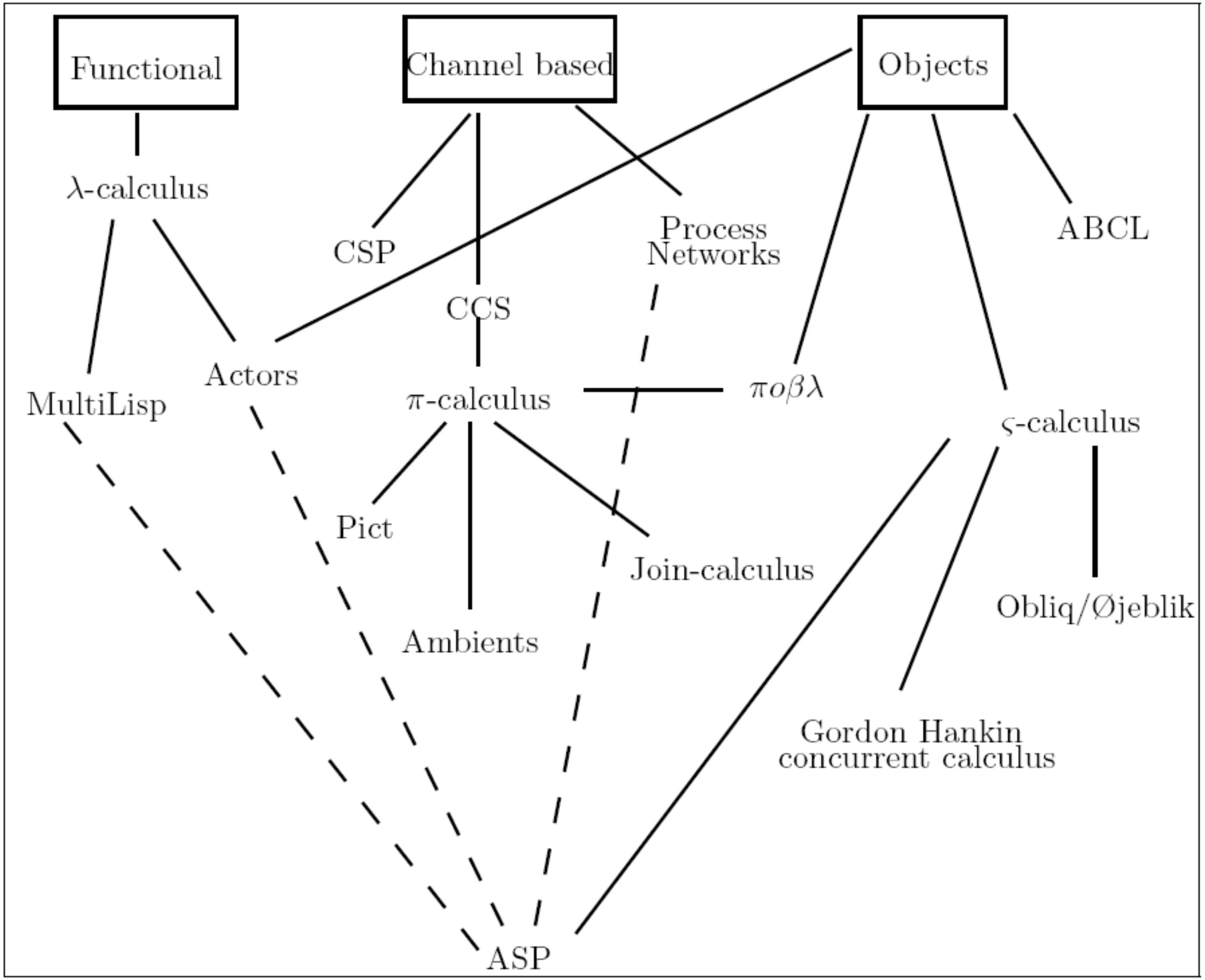
The different semantic families

- Denotational semantics
  - mathematical model, high level, abstract
- Axiomatic semantics
  - provides the language with a theory for proving properties / assertions of programs
- Operational semantics
  - computation of the successive states of an abstract machine
  - used to build evaluators, simulators.

What about concurrency and communication?

- Different timing (synchronous/asynchronous …)
- Different programming models (what is the unit of concurrency? What is sufficient to characterize an execution?...?)
- Interaction between communication/ concurrency/shared memory!

Through CCS, this course is a simple study of synchronous communications


Functional
Channel based
Objects
λ-calculus
CSP
Process Networks
ABCL
CCS
MultiLisp
Actors
π-calculus
πoβλ
ς-calculus
Pict
Join-calculus
Obliq/Øjeblik
Ambients
Gordon Hankin concurrent calculus
ASP

SEMANTICS

Operational Semantics

- Describes the computation
- States and configuration of an abstract machine:
  - Stack, memory state, registers, heap...
- Abstract machine transformation steps
- Transitions: current state -> next state
- Several different operational semantics

Natural Semantics : big steps (Kahn 1986)

- Defines the results of evaluation.
- Direct relation from programs to results

env |- prog => result

  - env: binds variables to values
  - result: value given by the execution of prog

Reduction Semantics : small steps

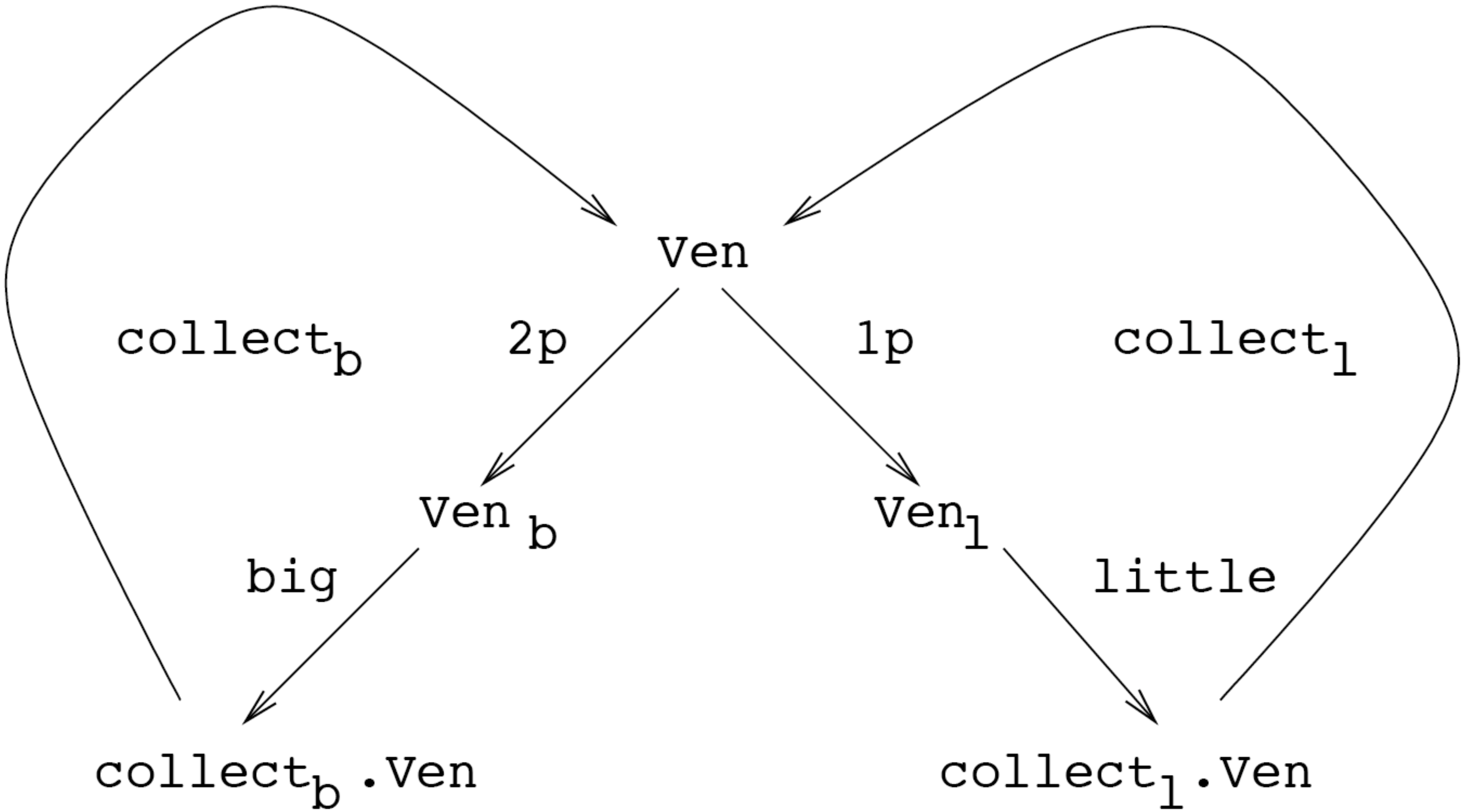
describes **each elementary step** of the evaluation

- **rewriting relation** : reduction of program terms
- **stepwise reduction**: <prog, s> -> <prog', s '>
  - infinitely, or until reaching a normal form.

Labelled Transition Systems (LTS)

- Basic model for representing reactive, concurrent, parallel, communicating systems.
- Definition:
  - < S, s0, L, T>
  - S = set of states
  - $S0 \in S$ = initial state
  - L = set of labels (events, communication actions, etc)
  - $T \subseteq S \times L \times S$ = set of transitions
  - Notation: $s1 \xrightarrow{a} s2 = (s1, a, s2) \in T$

An example



Ven
collect_b
2p
1p
collect_l
Ven_b
Ven_l
big
little
collect_b.Ven
collect_l.Ven

Deduction Rules

$$\frac{P \rightarrow Q \quad P}{Q}$$

$$\frac{P}{P \vee Q} \qquad \frac{Q}{P \vee Q}$$

CCS – SYNTAX AND SEMANTICS

CCS syntax

- Channel names: a, b, c , . . .
- Co-names: $\bar{a}, \bar{b}, \bar{c}, \ldots$
- Silent action: τ
- Actions: $\mu ::= a \mid \bar{a} \mid \tau$
- Processes:

P, Q	$::=$	0	inaction
	$\mid$	$\mu.P$	prefix
	$\mid$	$P \mid Q$	parallel
	$\mid$	$P + Q$	(external) choice
	$\mid$	$(\nu a)P$	restriction
	$\mid$	$\mathrm{rec}_K P$	process P with definition $K = P$
	$\mid$	K	(defined) process name

A tiny example

$$rec_{C1}(Tick.C1)$$

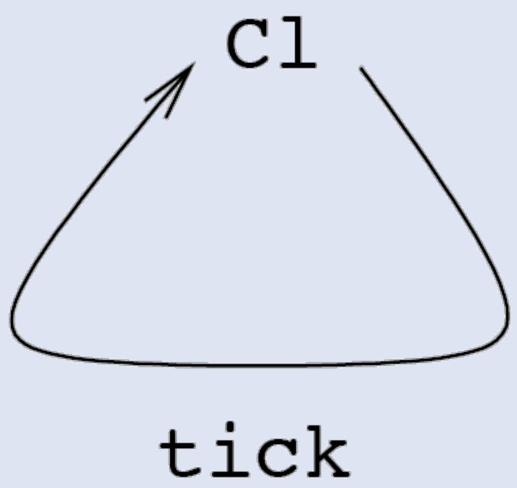


Figure: The transition graph for C1

Labelled graph

- vertices: process expressions
- labelled edges: transitions
- Each derivable transition of a vertex is depicted
- Abstract from the derivations of transitions

CCS : behavioural semantics (1)
Operators and rules

- Action prefix:

$$\frac{}{\mu.P \xrightarrow{\mu} P}$$

- Communication:

$$\frac{P \xrightarrow{a} P' \qquad Q \xrightarrow{\bar{a}} Q'}{P|Q \xrightarrow{\tau} P'|Q'}$$

- Parallelism

$$\frac{P \xrightarrow{\mu} P'}{P|Q \xrightarrow{\mu} P'|Q} \qquad \frac{Q \xrightarrow{\mu} Q'}{P|Q \xrightarrow{\mu} P|Q'}$$

CCS : behavioural semantics (2)
Operators and rules

- Non-deterministic choice

$$\frac{Q \xrightarrow{\mu} Q'}{P+Q \xrightarrow{\mu} Q'} \qquad \frac{P \xrightarrow{\mu} P'}{P+Q \xrightarrow{\mu} P'}$$

- Scope restriction

$$\frac{P \xrightarrow{\mu} P' \qquad \mu \neq a, \bar{a}}{(\nu a)P \xrightarrow{\mu} (\nu a)P'}$$

- Recursive definition

$$\frac{P[\text{rec}_K P / K] \xrightarrow{\mu} P'}{\text{rec}_K P \xrightarrow{\mu} P'}$$

Derivations
(construction of each transition step)

$$\dfrac{\dfrac{\dfrac{}{a.P \xrightarrow{\;a\;} P}\ \text{Prefix}}{a.P \mid Q \xrightarrow{\;a\;} P \mid Q}\ \text{Par-L} \qquad \dfrac{}{\bar{a}\,.R \xrightarrow{\;\bar{a}\;} R}\ \text{Prefix}}{(a.P \mid Q) \mid \bar{a}.R \xrightarrow{\;\tau\;} (P \mid Q) \mid R}\ \text{Par-2}$$

Par-2(Par_L(Prefix), Prefix)

Another one :

Par-L(Par_L(Prefix))

One amongst 3 possible derivations

$$(a.P \mid Q) \mid \bar{a}.R \xrightarrow{\;\;\;\;a\;\;\;\;} (P \mid Q) \mid \bar{a}.R$$

EQUIVALENCES

Behavioural Equivalences

- Intuition:
  - Same possible sequences of observable actions
  - Finite / infinite sequences
  - Various refinements of the concept of observation
- Definition: Trace Equivalence

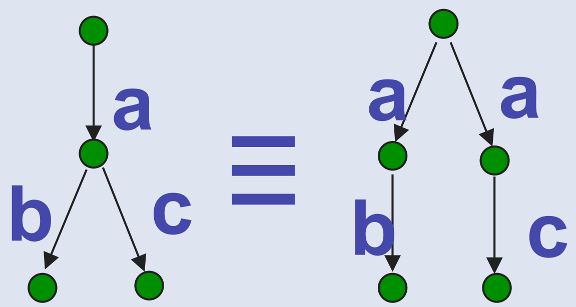
  For a LTS (S, s0, L, T) its Trace language ***T*** is the set of finite sequences {(t = t_1, …, t_n such that $\exists s_0,\ldots,s_n \in S^{n+1}$, and $(s_{n-1},t_n,s_n) \in T$}

  Two LTSs are Trace equivalent iff their Trace languages are equal.

  Corresponding Ordering: Trace inclusion

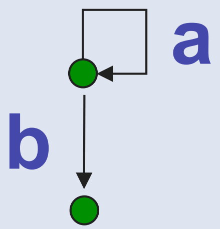
Trace Languages, Examples

- Those 2 systems are trace equivalent:



T = {(), (a), (a,b), (a,c)}

- A trace language can be an infinite set:



T = {(), (a), (a,a), (a,…,a),…
(a,b), (a,a,b), (a,a,…,a,b),
…}

Bisimulation

- ## Behavioural Equivalence
  - non distinguishable states by observation:

    two states are equivalent if for all possible transitions labelled by the same action, there exist equivalent resulting states.

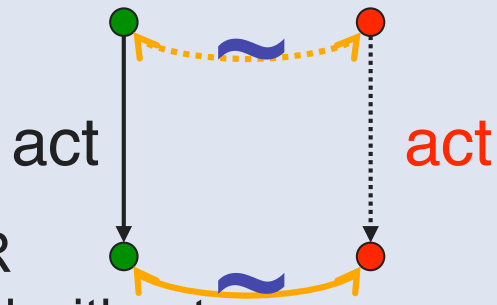
- ## Bisimulations

  R ⊆ SxS is a simulation iff
  - It is a equivalence relation
  - ∀(p,q) ∈ R,

    (p,l,p’) ∈ T => ∃ q’/ (q,l,q’) ∈ T and (p’,q’) ∈ R
  - R is a **bisimulation** if the same condition hold with q too:

    ∀(p,q) ∈ R,

    (q,l,q’) ∈ T => ∃ q’/ (q,l,q’) ∈ T and (p’,q’) ∈ R



- ## ~ is the coarsest bisimulation

  2 LTS are bisimilar iff their initial states are in ~

  quotients = canonical normal forms

Transitivity

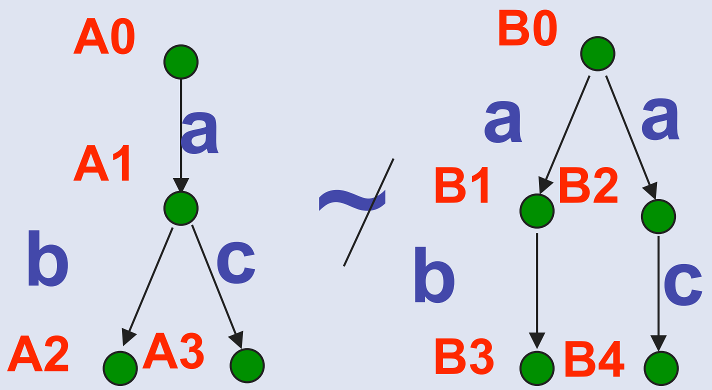
- If **R**, **S** are bisimulations, then so is their composition

RS = {(P, P') | ∃ Q. P **R** Q and Q **S** P'}

- In particular, ~~ ⊆ ~, i.e., bisimilarity is transitive.

Bisimulation

- More precise than trace equivalence :



No state in B is equivalent to A1

- Preserves deadlock properties.
- Can be built by adding elements in the equivalence relation
- Coinductive definition (biggest set verifying …)

Bisimulation

- Congruence laws:

  P1~P2 => a.P1 ~ a:.2 (∀ P1,P2,a)

  P1~P2, Q1~Q2 => P1+Q1 ~ P2+Q2

  P1~P2, Q1~Q2 => P1|Q1 ~ P2|Q2

  Etc…

- ~ is a congruence for all CCS operators :

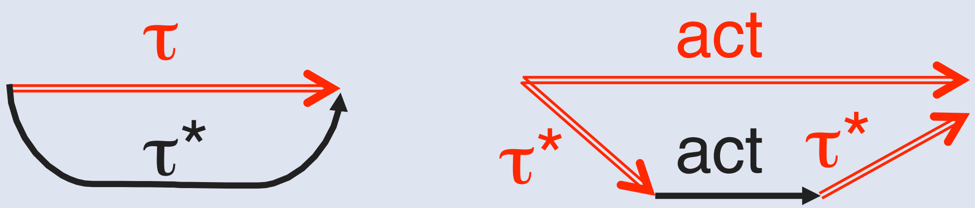
  for any CCS context C[.], C[P] ~ C[Q] <=> P~Q

  Basis for compositional proof methods

- Maximal trace is not an equivalence

Observational Equivalences

- Weak bisimulation
  - Abstraction: hidden actions
  - allows for arbitrary many internal actions $\overset{\mu}{\Rightarrow}$



Weak bisimulation

- The following def is a tractable version of weak bisimulation:

A weak bisimulation is a relation R *such that*

$P\ \mathbf{R}\ Q \Rightarrow \forall \mu, P, P'\ (P \rightarrow P' \overset{\mu}{\Rightarrow} \exists Q'.\ Q \overset{\mu}{\Rightarrow} Q'$ *and* $P'\ \mathbf{R}\ Q')$

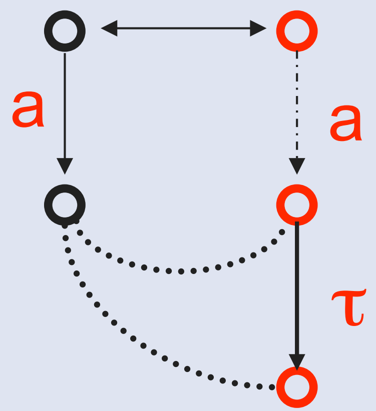
and conversely

- Note the dissymetry between the use of →on the left and of ⇒ on the right
- Two processes are *weakly bisimilar* if (notation $P \approx Q$) *if there exists a* weak bisimulation ***R*** *such that* $P\ \mathbf{R}\ Q$.

Branching bisimulation

- only staying in equivalent states

Still existence of a canonical minimal automata
Computation is polynomial



ADDITIONAL NOTATIONS AND CONSTRUCTS

Alternative Notations

- $\stackrel{\text{def}}{=}$

$$rec_{C1}(Tick.C1) \Leftrightarrow \mathtt{C1} \stackrel{\text{def}}{=} \mathtt{tick.C1}$$

a little more complex for several definitions

-> exercise?

- Input/output: a=?a ; $\overline{a}$ = !a
- | or ||

Extension: Parameterized actions

- input of data at port a, a(x).E
- a(x) binds free occurrences of x in E .
- Port a represents {a(v) : v $\in$ D } where D is a family of data values
- Output of data at port a, $\overline{a}$(e).E where e is a data expression.
- Transition Rules: depend on extra machinery for expression evaluation. Let Val(e) be data value in D (if there is one) to which e evaluates
- R (in) a(x).E $\xrightarrow{a(v)}$ E {v /x } if v $\in$ D where {v /x } is substitution
- R (out) $\overline{a}$(e).E $\xrightarrow{\overline{a}(v)}$ E if Val(e) = v

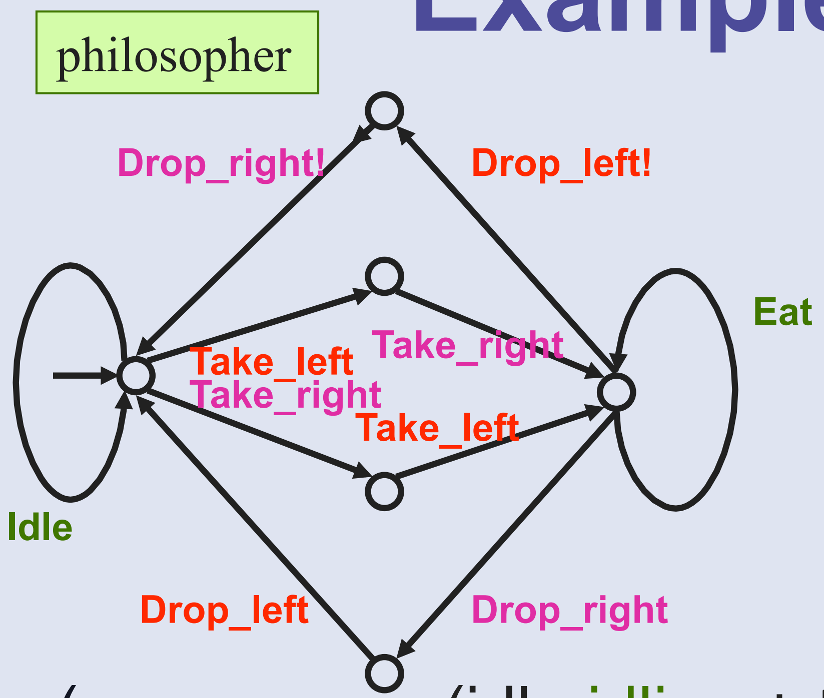
Example: a register

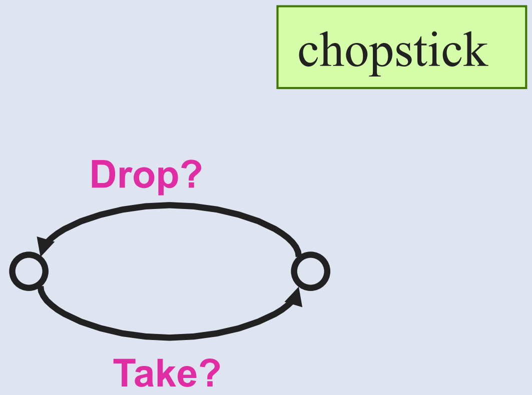
$$Reg_i = \overline{read}(i).Reg_i + write(x).Reg_x$$

$$\frac{\dfrac{Reg_5 \xrightarrow{write(3)} Reg_3}{\overline{read}(5).Reg_5 + write(x).Reg_x \xrightarrow{write(3)} Reg_3}}{write(x).Reg_x \xrightarrow{write(3)} Reg_3}$$

EXAMPLES

Example: dining philosophers





(rec$_{\text{idling,eating}}$. (idle.idling + take_left.take_right.eating + take_right.take_left.eating,

eat.eating + drop_left.drop_right.idling + drop_right.drop_left.idling)

Deadlock or not ?

Mutual exclusion ?

(trivial) example: Milner's Scheduler

- Processes iteratively start and finish executing tasks (one task per process)
- Task starts are cyclically ordered

cycler = $\overline{\alpha}$.start.(β.0 || end.cycler)

scheduler_3 = local α1, α2, α3 in

([α1/ α, α2/β, start1/start, end1/end] cycler

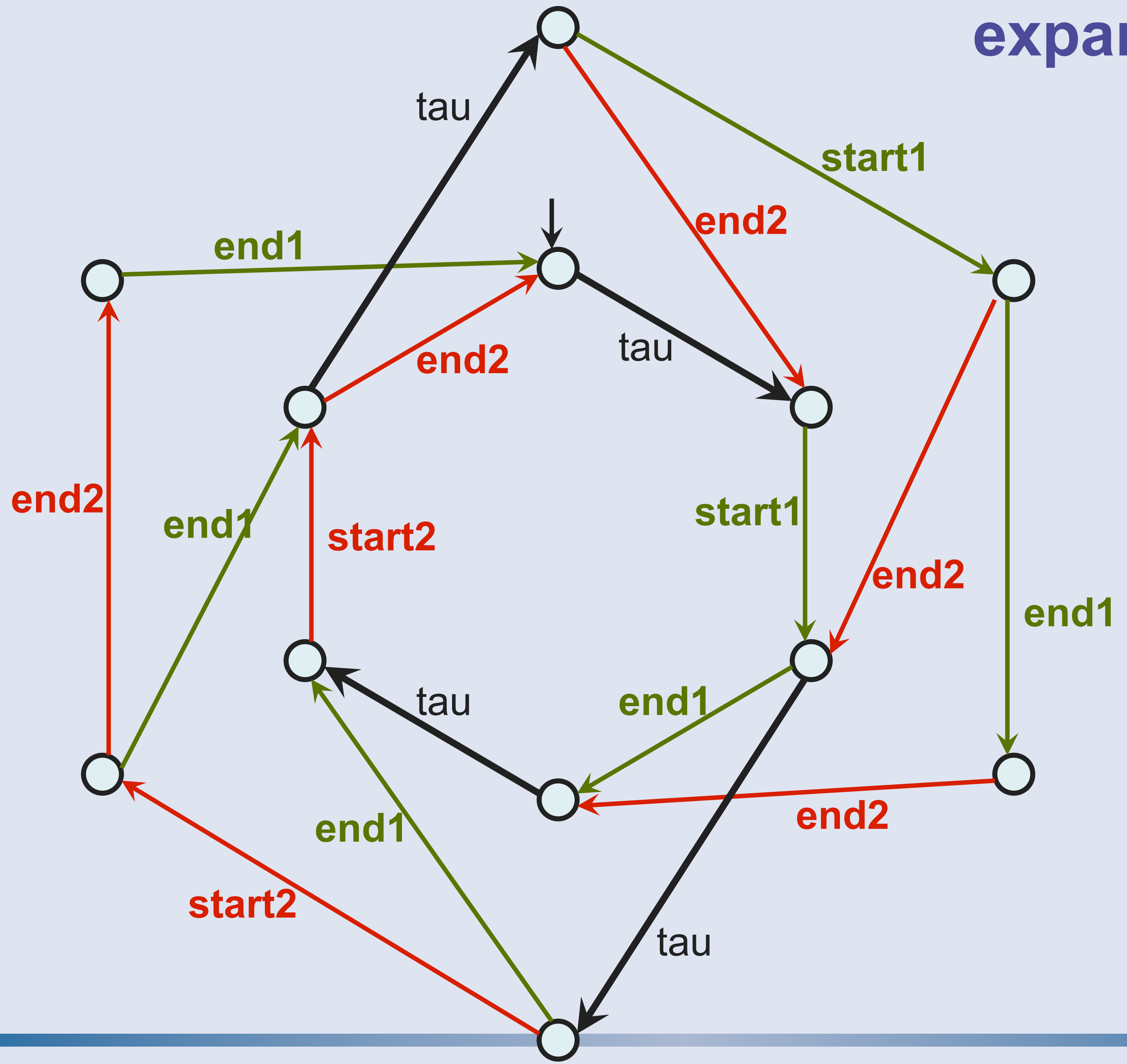
|| [α2/ α, α3/β, start2/start, end2/end] cycler

|| [α3/ α, α1/β, start3/start, end3/end] cycler

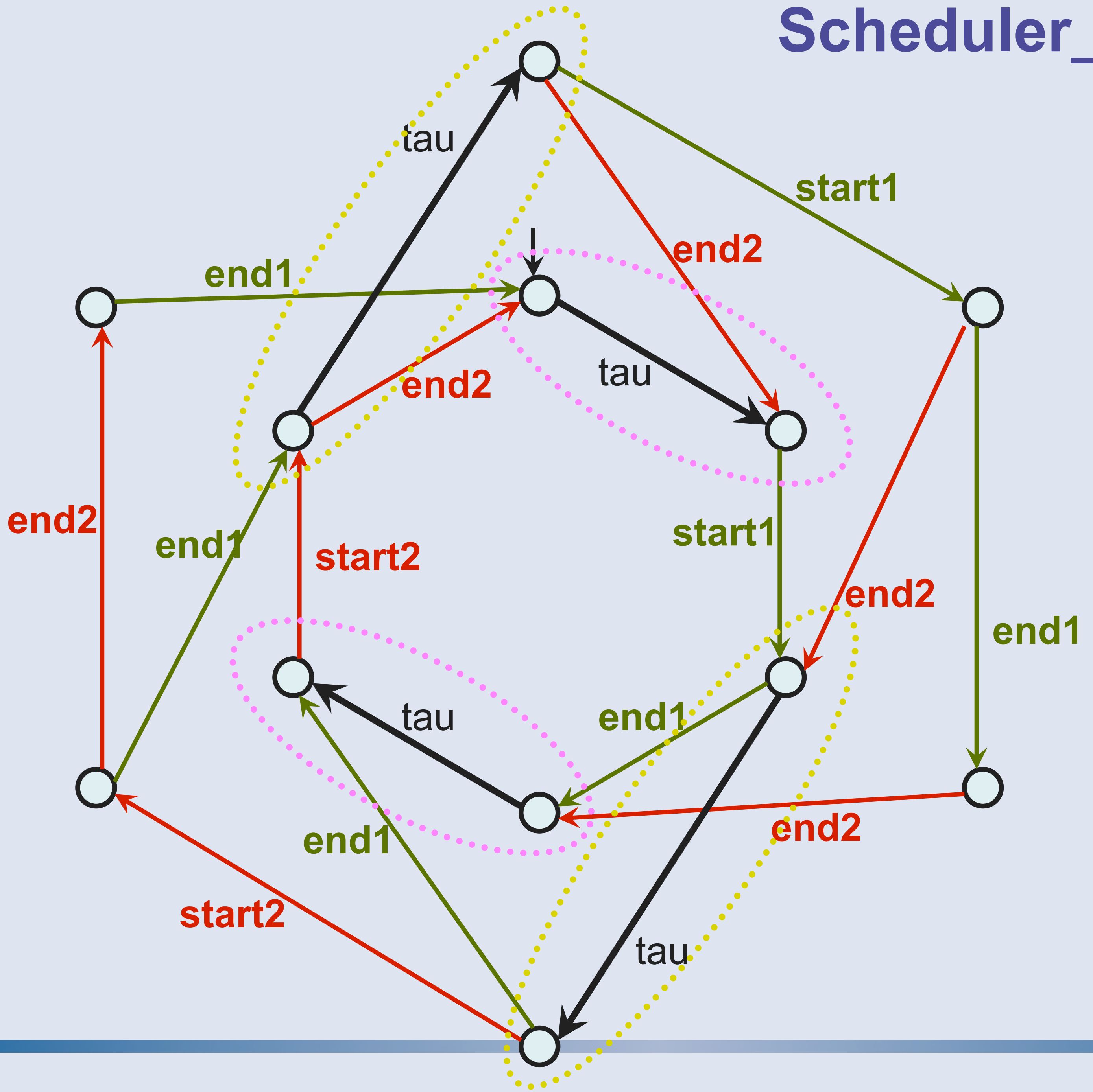
|| α1.0)

vérification des propriétés ?

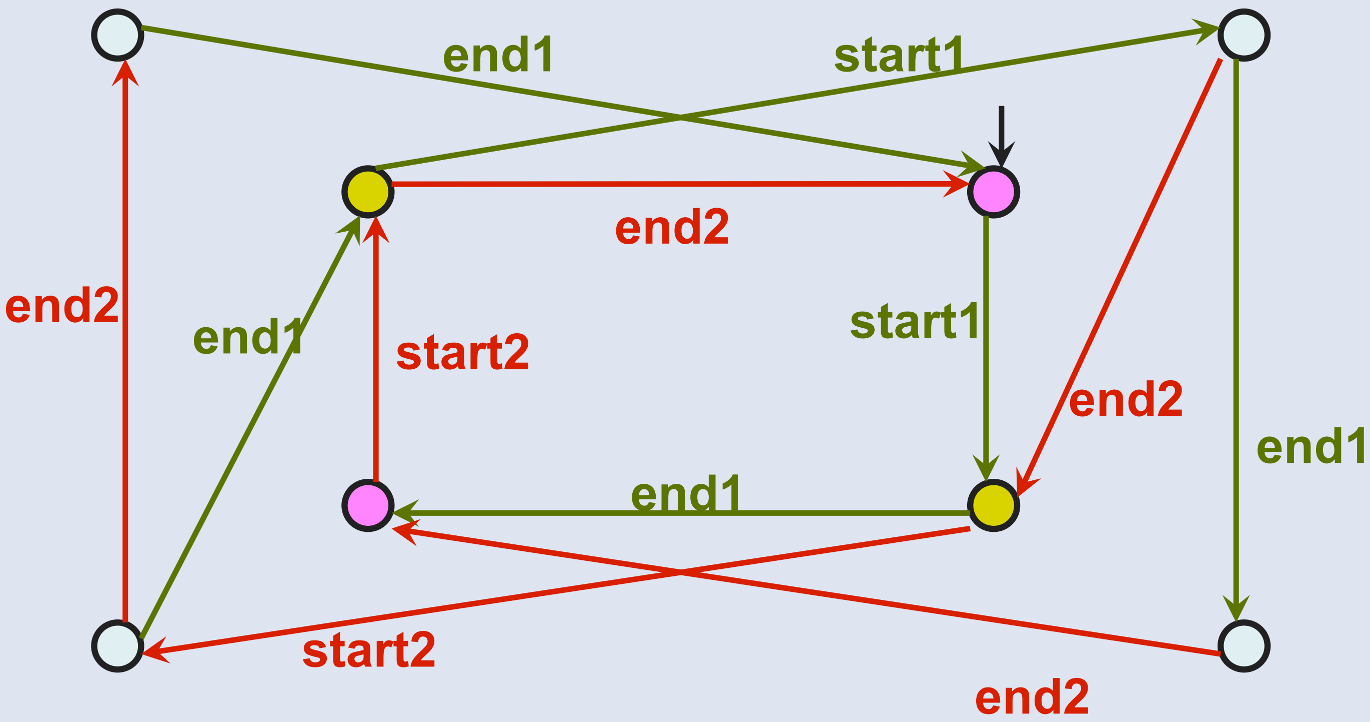
Scheduler_2 expanded



Scheduler_2 reduced



Scheduler_2 reduced

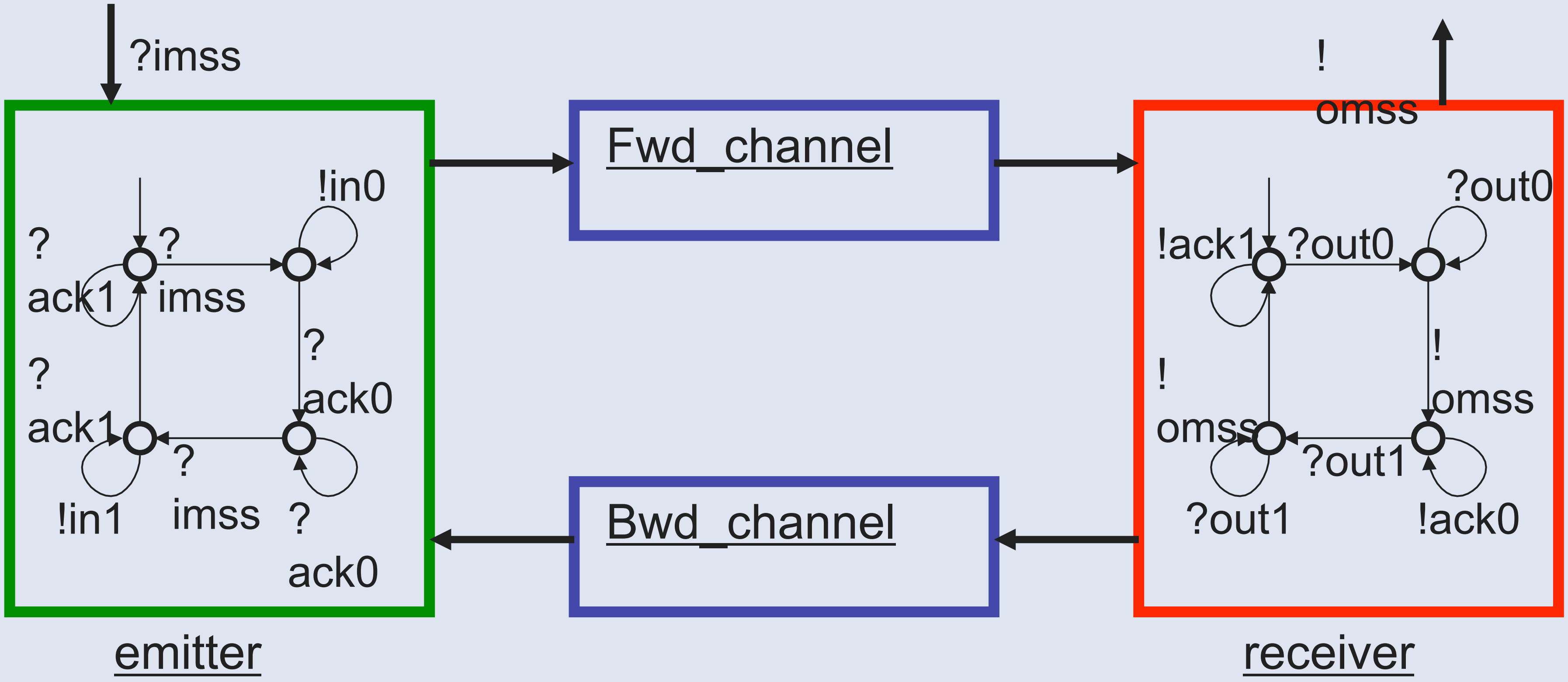


CONCLUSION

- A synchronous communication language
- A (complex but) efficient notion of equivalence on processes
- What is missing?
  - Channel communication (like in pi-calculus) -> much more complex
  - No computational construct by nature

EXERCISES

Example: Alternated Bit Protocol



Write in CCS ?

Hypotheses: channels can loose messages

Requirement:

the protocol ensures no loss of messages

Example: Alternated Bit Protocol (2)

- **emitter =**

```
let rec {em0 = ?ack1 :em0 + ?imss:em1
       and em1 = !in0 :em1 + ?ack0 :em2
       and em2 = ?ack0 :em2 + ?imss :em3
       and em3 = !in1 :em3 + ?ack1 :em0
  }
       in em0
```

- **ABP =** local {in0, in1, out0, out1, ack0, ack1, …}
  in emitter || Fwd_channel || Bwd_channel || receiver

Example: Alternated Bit Protocol (3)

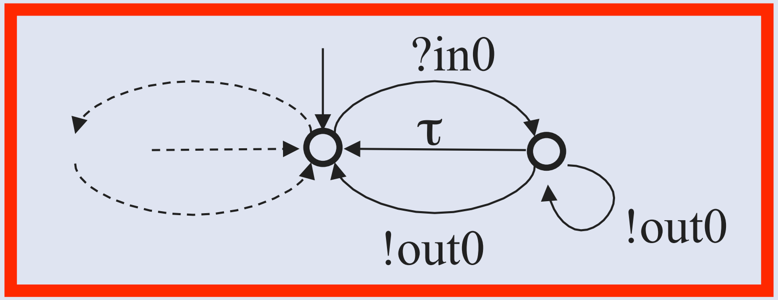
Channels that loose and duplicate messages (in0 and in1) but preserve their order ?

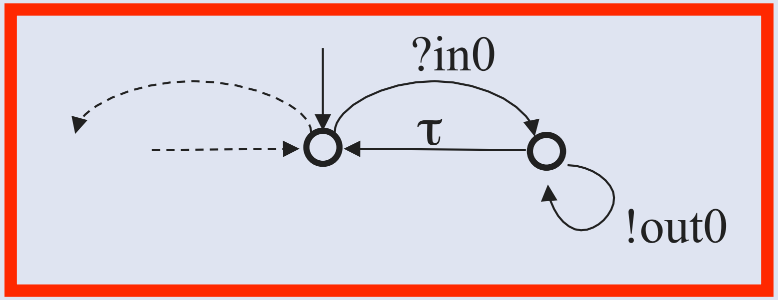
- Exercise :
  1) Draw an LTS describing the loosy channel behaviour
  2) Write the same description in CCS

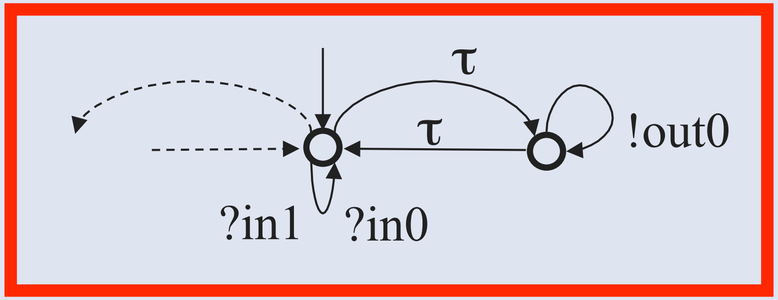
Exercise 2

- $\text{rec}_K \mathit{coin}.(\mathit{coffee}.\overline{\mathit{ccup}}.K + \mathit{tea}.\overline{\mathit{tcup}}.K)$
- $\mathit{coin}.\text{rec}_K(\mathit{coffee}.\overline{\mathit{ccup}}.\mathit{coin}.K + \mathit{tea}.\overline{\mathit{tcup}}.\mathit{coin}.K)$
- $\text{rec}_K(\mathit{coin}.\mathit{coffee}.\overline{\mathit{ccup}}.K + \mathit{coin}.\mathit{tea}.\overline{\mathit{tcup}}.K)$
- Question: which of these machines can we safely consider equivalent?
- Note that these machines have all the same traces.

Exercice 3 : Bisimulations







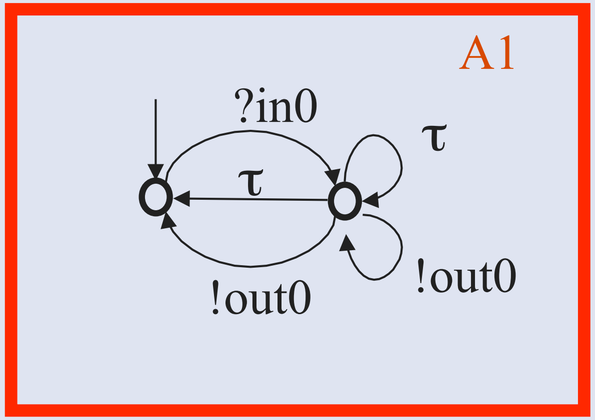
Are those 3 LTSs equivalent by:

- Strong bisimulation?

- Weak bisimulation ?

In each case, give a proof.

Exercice 3 : Bisimulation



- Exercice :
  1) Compute the strong minimal automaton for A1.
  2) Compute the weak minimal automaton for A1.

Exercise 5

- Compare the construct $\overset{\text{def}}{=}$ and rec_K :
  1. Let us start by a simple pair of processes

$$A \overset{def}{=} \bar{a}.A + b.B$$

$$B \overset{def}{=} a.A$$

  2. Suppose rec can accept several variables: rec (K=P,L=Q) express the same term
  3. Is it possible to express the same thing with a single variable K? Here are some possible hints:
     - Define a recursive process All that contains A and B and can trigger each of them by the reception of a message on channel cA or cB
     - (we suppose cA and cB cannot be used elsewhere)
     - What kind of equivalence between the two expressions do you have?

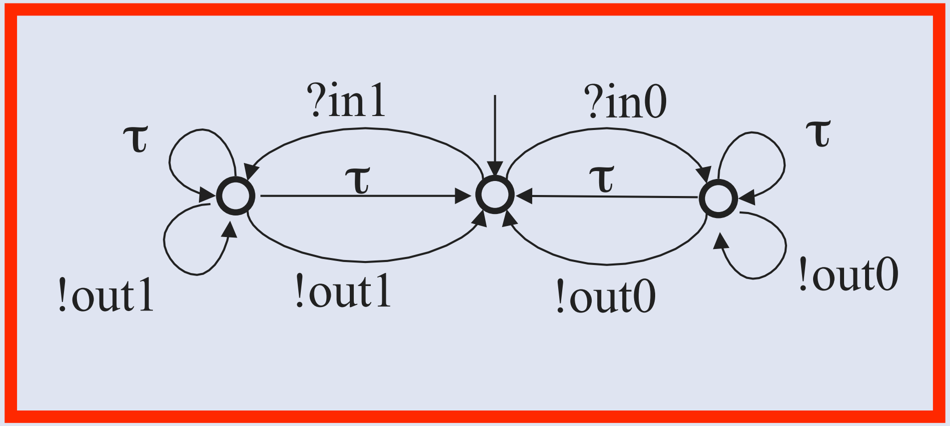
CORRECTION

Exercice: Alternated Bit Protocol

Correction (1):

Channels that loose and duplicate messages (in0 and in1) but preserve their order ?

1) Draw an automaton describing the loosy channel behaviour



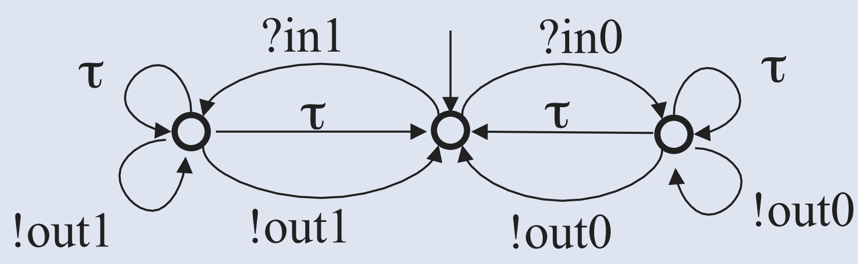
- It is a symmetric system, receiving ?in0 and ?in1 messages, then delivering 0 , 1 or more times the corresponding !out0 or !out1 message.
- On each side (bit 0 or 1), the initial state has a single transition for the reception.
- In the next state, it can either : return silently to the initial state (= lose the message), deliver the message and return to the initial state (exactly one delivery), or deliver the message and stay in the same state (thus enabling duplication).

Exercice: Alternated Bit Protocol

Correction (2):

Channels that loose and duplicate messages (in0 and in1) but preserve their order ?

2) Write it in CCS



- **Lousy channel** =

let rec {ch0 = ?in0 :ch1 + ?in1:ch2

and ch1 = τ :ch1 + τ :ch0 + !out0 :ch1 + !out0 :ch0

and ch2 = τ :ch2 + τ :ch0 + !out0 :ch2 + !out0 :ch0

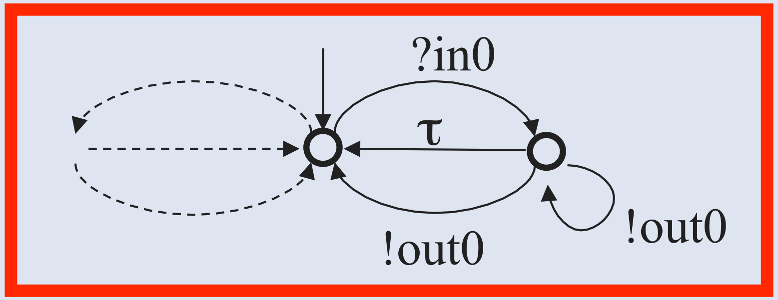
}

in ch0

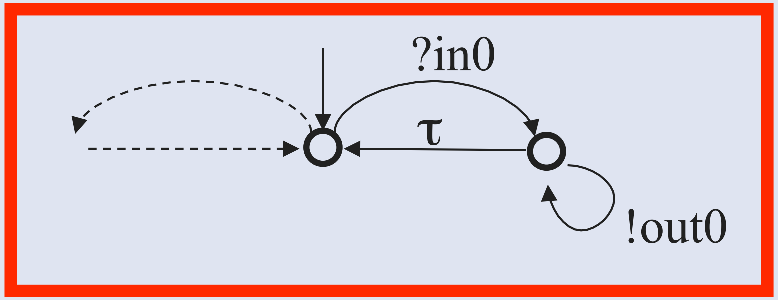
Exercice: Alternated Bit Protocol

Correction (3):

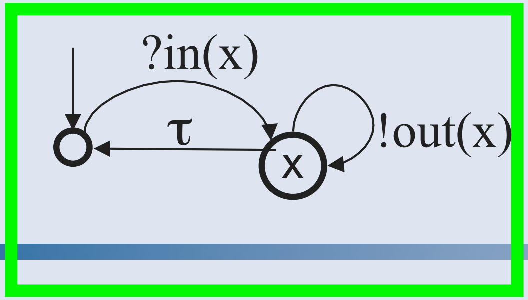
Channels that loose and duplicate messages (in0 and in1) but preserve their order ?



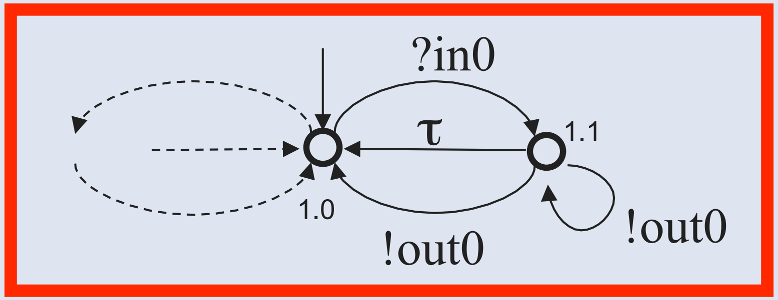
Other Solutions:

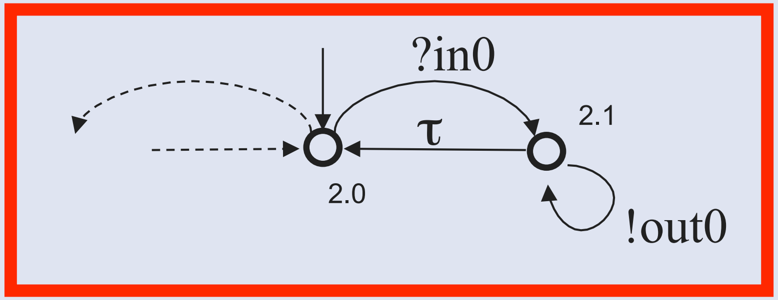


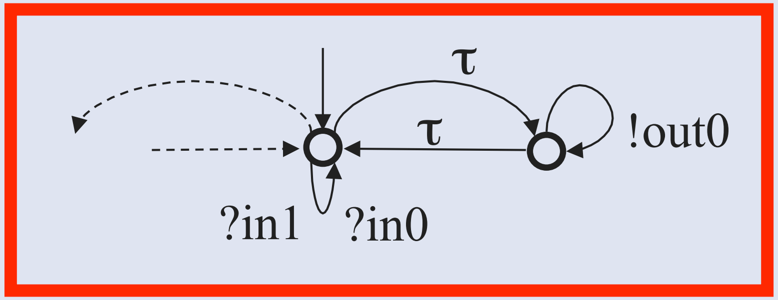
More generally, parameterized model :



Exercice 2 : Bisimulations







Are those 3 LTSs equivalent by:

- Strong bisimulation?

NO ! Need find non equivalent states. E.g. counter example for $1 \neq 2$:

States 1.0 and 1.1 are different because 1.0 can do ?in0 and 1.1 cannot.

Then 1.1 and 2.1 are different because 1.1 can do !out0 -> 1.0, while no 2.1 !out0 transitions can go to a state equivalent to 1.0.

- Weak bisimulation ?

YES. Exhibit a partition of equivalent states:

1={1.0,2.0}, 2={1.1, 2.1}

Check all possible (τ*aτ*) transitions:

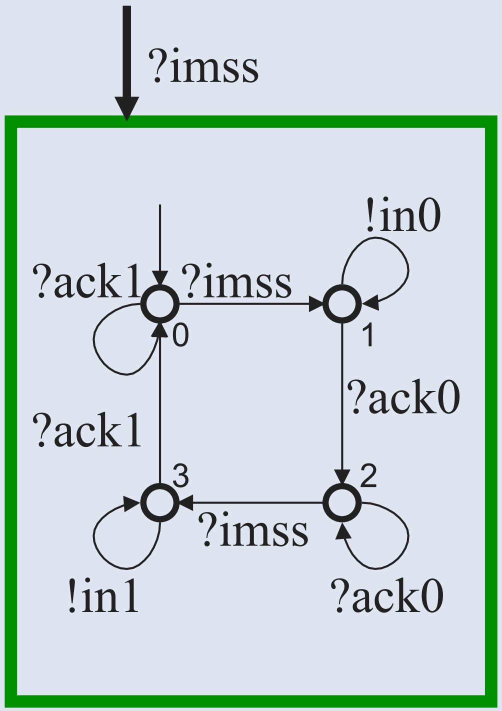
1 - !in0 -> 2, … , 2 - !out0.τ* -> 1

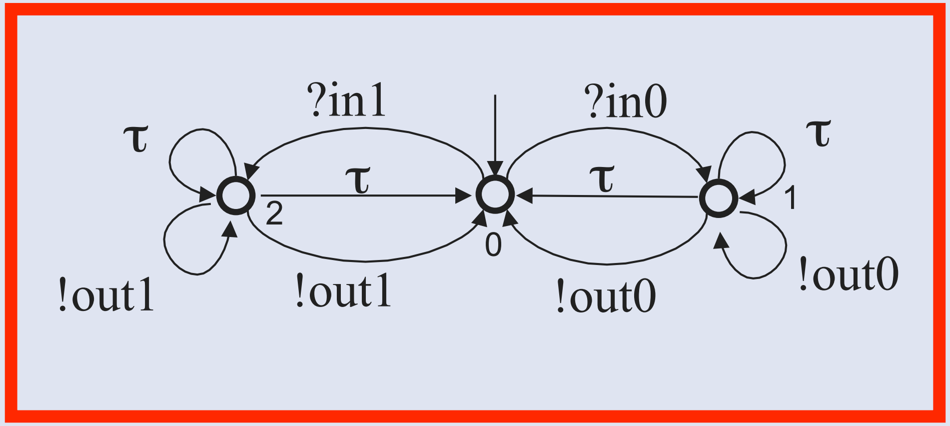
Remark: this transition set defines the minimal representant modulo weak bisimulation…

Exercice 4 : Produit synchronisé

Compute the synchronized product of the LTS representing the ABP emitter with the (forward) Channel:

local {in0, in1} in
(Emitter || Channel)

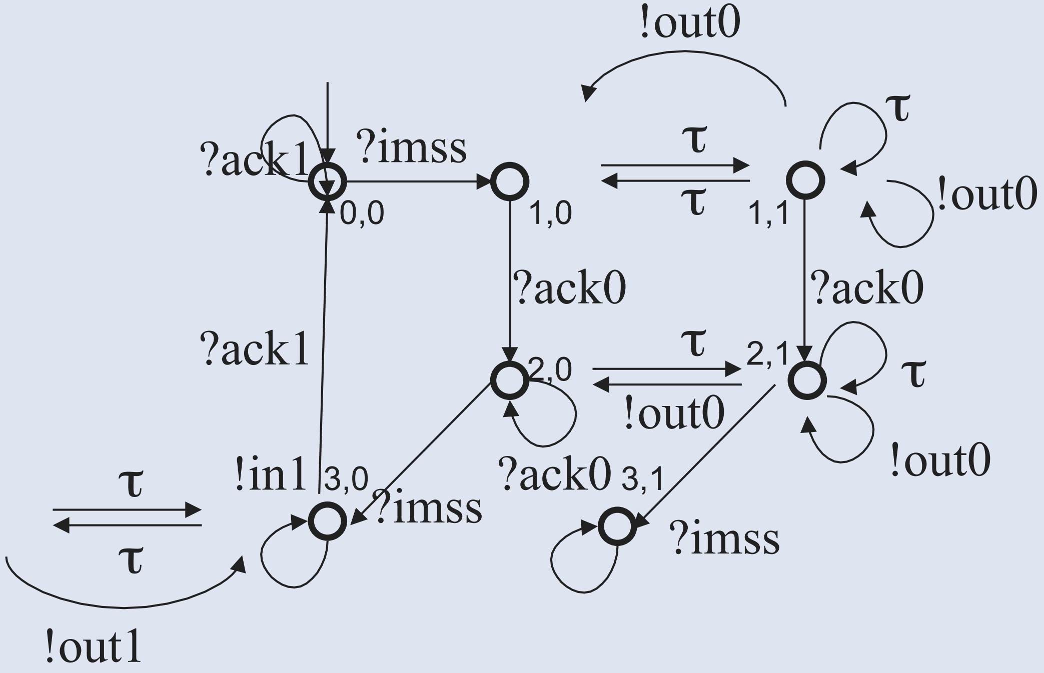




Exercice 4 : Produit synchronisé

Correction ? partially…

local {in0, in1} in
(Emitter || Channel)



Exercice 4 : Produit synchronisé

Correction ? Tool generated LTS…