Reactive and Event Based Systems: Part II

Formal process languages as foundation for programming languages

December 13th, 2021

Thomas Hildebrandt,
Software, Data, People & Society
Datalogisk Institut Københavns Universitet

KØBENHAVNS UNIVERSITET

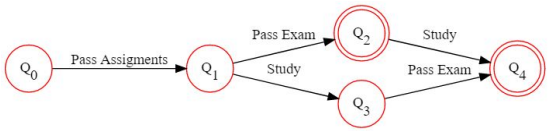
Overview of Part II: Formal process languages as foundation for programming languages for reactive and event based systems

- Week 4 (Dec 13th): Formal process languages for distributed, reactive and event based systems & brief intro to Jolie
- Week 4 (Dec 17th): Jolie
- Week 5 (Dec 20th): Choreographies and Services in the Jolie programming language: Hand-in 2

Formal process languages for Distributed, reactive and event based systems ?

Automata/State-based models

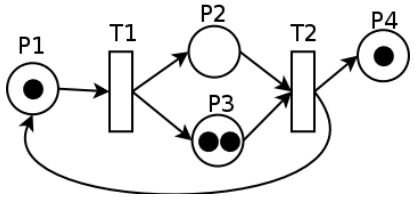
Finite automata (McCulloch & Pitts'43)



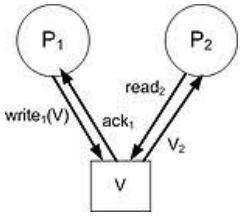
and regular languages ()

PA.(PE.S* + S.PE)

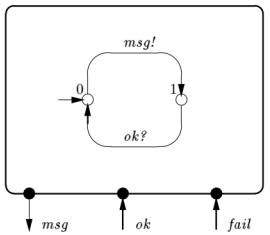
Petri Net (C. A. Petri '62)



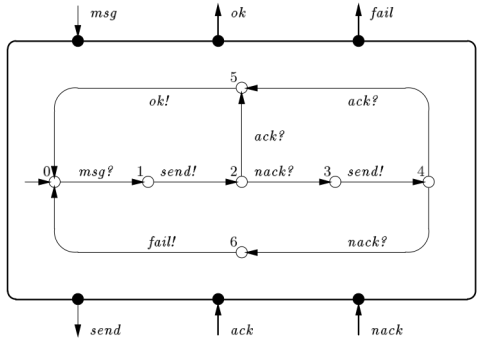
I/O Automata (N. Lynch, M. Tuttle '87)



Interface Automata (L. Alfaro, T. Henzinger '91)



(a) Interface automaton *User*



(b) Interface automaton *Comp*

Used to give formal specifications of concurrent and distributed computational systems
- and semantics of programming languages

Formal process languages for Distributed, reactive and event based systems ?

Process models & calculi

Programming languages & toolkits

Lambda-calculus (30ties A. Church)

$$(\lambda x.\, t) \qquad (\lambda x.\, t)s \rightarrow t[x := s]$$

Formal process languages for Distributed, reactive and event based systems ?

Process models & calculi

Lambda-calculus (30ties A. Church)

$$(\lambda x.\, t) \qquad (\lambda x.\, t)s \rightarrow t[x := s]$$

Programming languages & toolkits

ML ‘73
Miranda ‘85

Haskell ‘90

Formal process languages for Distributed, reactive and event based systems ?

Process models & calculi

Lambda-calculus (30ties A. Church)

$$(\lambda x.\, t) \qquad (\lambda x.\, t)s \rightarrow t[x := s]$$

Programming languages & toolkits

Simula ‘67

ML ‘73
Miranda ‘85

Haskell ‘90

Formal process languages for Distributed, reactive and event based systems ?

Process models & calculi

Lambda-calculus (30ties A. Church)

$$(\lambda x.\, t) \qquad (\lambda x.\, t)s \rightarrow t[x := s]$$

Actor Model (Hewitt ' 73)
Asynchronous messages (send, get) & spawning new actors

Dataflow Network ('74)

Programming languages & toolkits

Simula '67

ML '73
Miranda '85

Haskell '90

Actors – Hewitt

Messages guaranteed to arrive eventually – but not in order send.

No shared state between Actors.

Addresses of Actors may be send in messages.

Hewit, Bishop, Steiger

IJCAI'73: Proceedings of the 3rd international joint conference on Artificial intelligence, August 1973 Pages 235–245

Artificial Intelligence

A Universal Modular ACTOR Formalism
for Artificial Intelligence

Carl Hewitt
Peter Bishop
Richard Steiger

Abstract

This paper proposes a modular ACTOR architecture and definitional method for artificial intelligence that is conceptually based on a single kind of object: actors [or, if you will, virtual processors, activation frames, or streams]. The formalism makes no presuppositions about the representation of primitive data structures and control structures. Such structures can be programmed, micro-coded, or hard wired 1n a uniform modular fashion. In fact it is impossible to determine whether a given object is "really" represented as a list, a vector, a hash table, a function, or a process. The architecture will efficiently run the coming generation of PLANNER-like artificial intelligence languages including those requiring a high degree of parallelism. The efficiency is gained without loss of programming generality because it only makes certain actors more efficient; it does not change their behavioral characteristics. The architecture is general with respect to control structure and does not have or need goto, interrupt, *or* semaphore primitives. The formalism achieves the goals that the disallowed constructs are intended to achieve by other more structured methods.

PLANNER Progress

"Programs should not only work,
but they should appear to work as well."
PDP-1X Dogma

Dataflow Network (Kahn, ⋯)

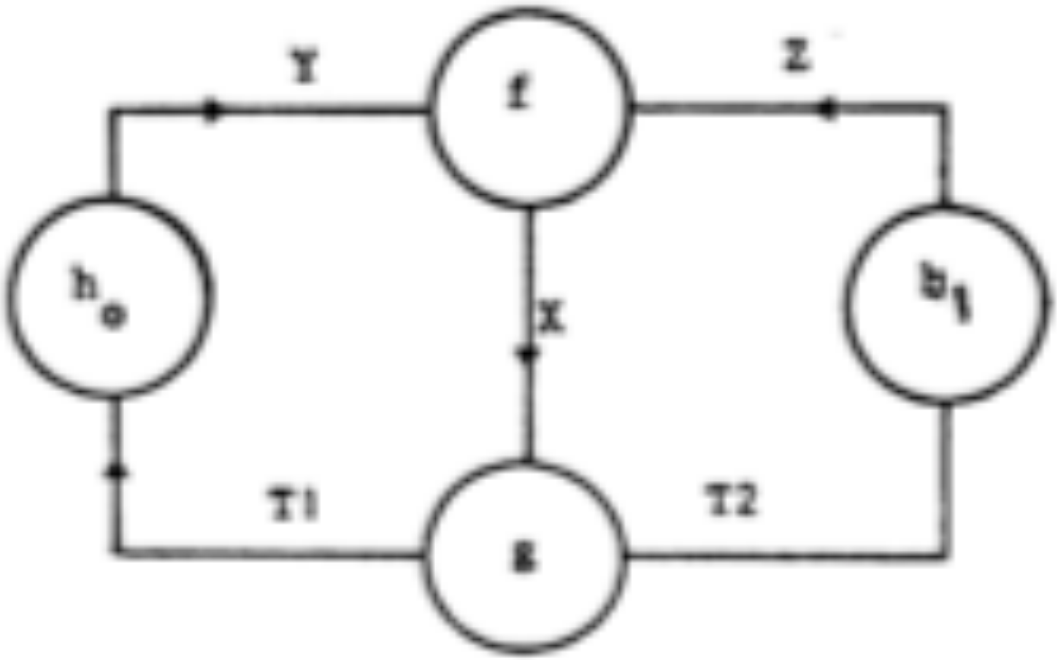
Messages guaranteed to arrive eventually in order send.

No shared state between Dataflow nodes

Fixed topology

Long open challenge to give relational compositional semantics to non-deterministic dataflow networks

Dataflow program (Kahn 1974)



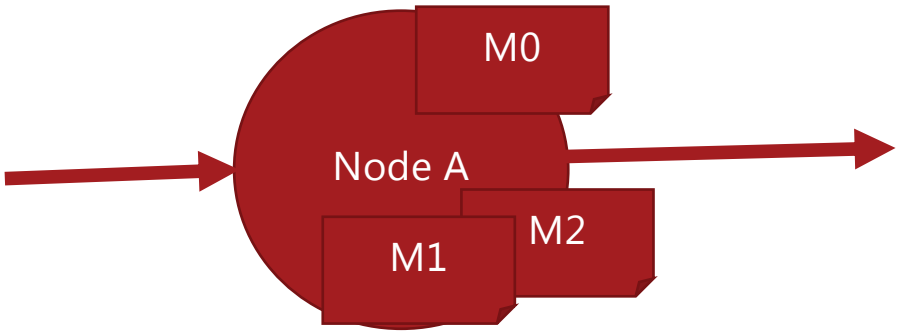

```
Begin
(1) Integer channel X, Y, Z, T1, T2 ;
(2) Process f(integer in U,V; integer out W) ;
    Begin integer I ; logical B ;
          B := true ;
          Repeat Begin
(4)         I := if B then wait(U) else wait(V) ;
(7)         print (I) ;
(5)         send I on W ;
            B := ¬B ;
            end ;
    End ;
  Process g(integer in U ; integer out V, W) ;
    Begin integer I ; logical B ;
      B := true ;
      Repeat Begin
        I := wait (U) ;
        if B then send I on V else send I on  W ;
        B := ¬B ;
        End ;
    End ;
(3) Process h(integer in U;integer out V; integer INIT);
    Begin integer I ;
      send INIT on V ;
      Repeat Begin
        I := wait(U) ;
        send I on V ;
        End ;
    End ;
  Comment : body of mainprogram ;
(6) f(Y,Z,X) par g(X,T1,T2) par h(T1,Y,0) par h(T2,Z,1
  End ;
```

Fig.1. Sample parallel program S.

Key point:
Semantics defined as continuous functions between streams

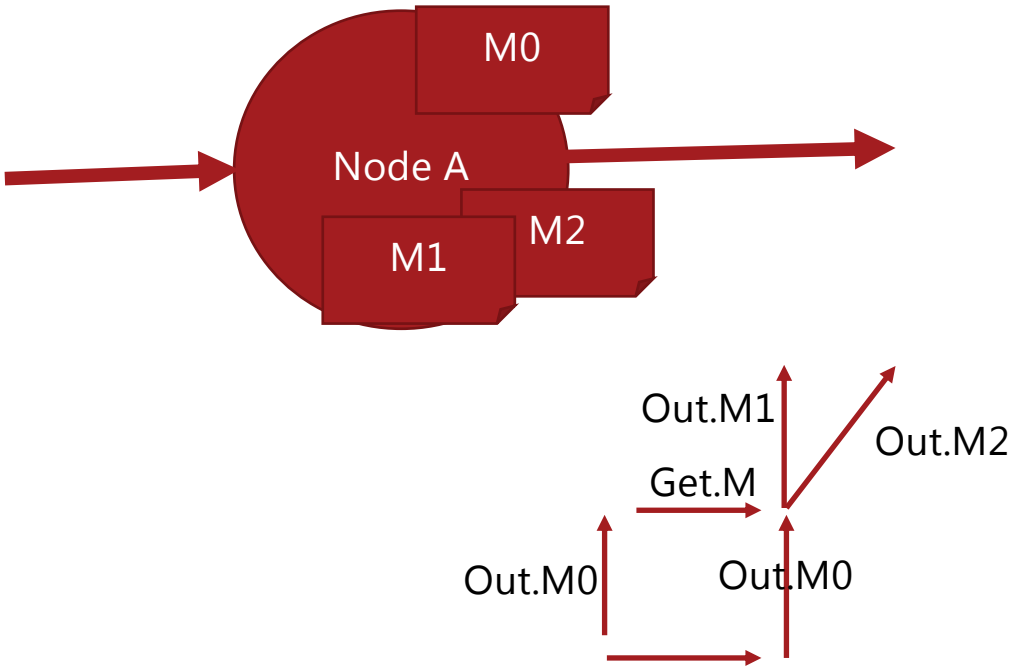
Non-deterministic Dataflow Network

After receiving a message and having output the message M0, then Node A may output either M1 or M2



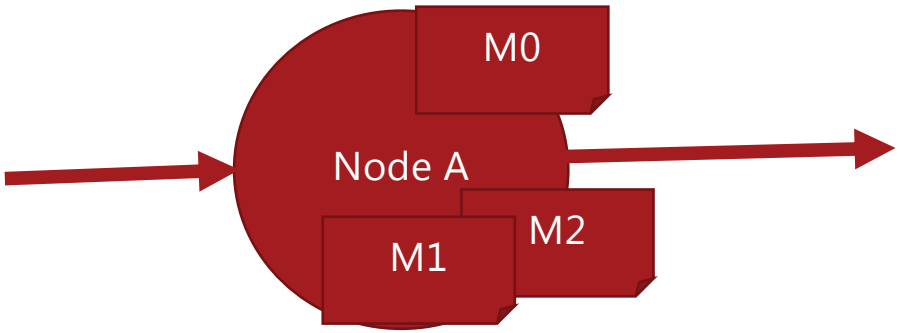
Non-deterministic Dataflow Network – Transition Semantics

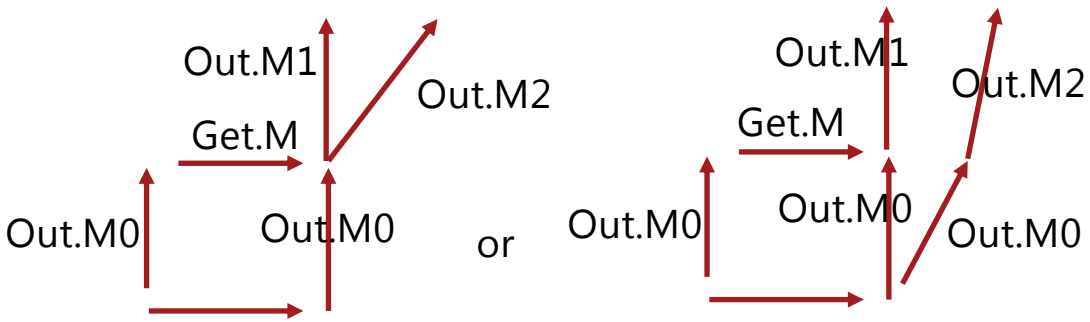
After receiving a message and having output the message M0, then Node A may output either M1 or M2



Non-deterministic Dataflow Network – branching points

After receiving a message and having output the message M0, then Node A may output either M1 or M2

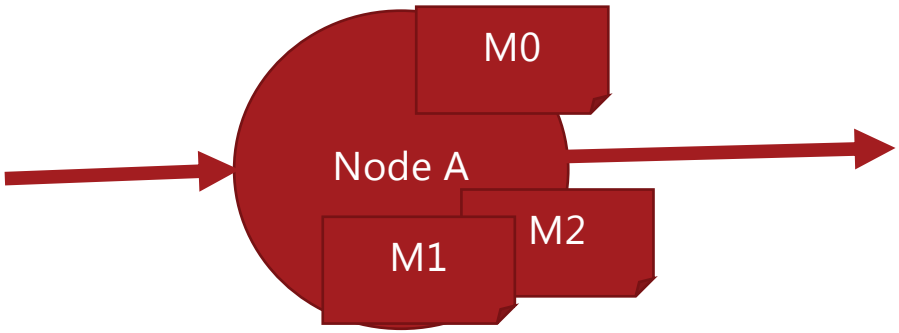


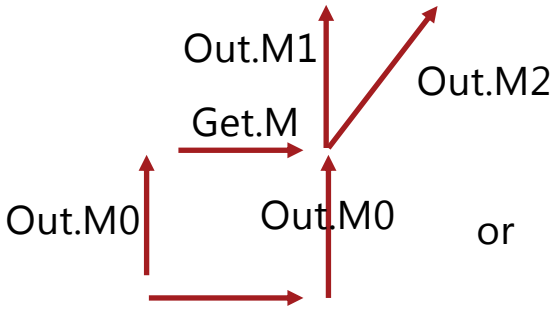


In the 2nd version, Node A decide salready before outputting M0 if it should output M1 or M2

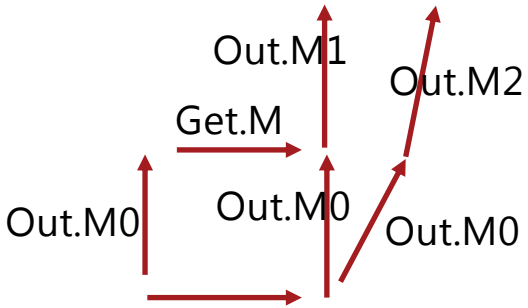
Non-deterministic Dataflow Network – input/output relations

After receiving a message and having output the message M0, then Node A may output either M1 or M2





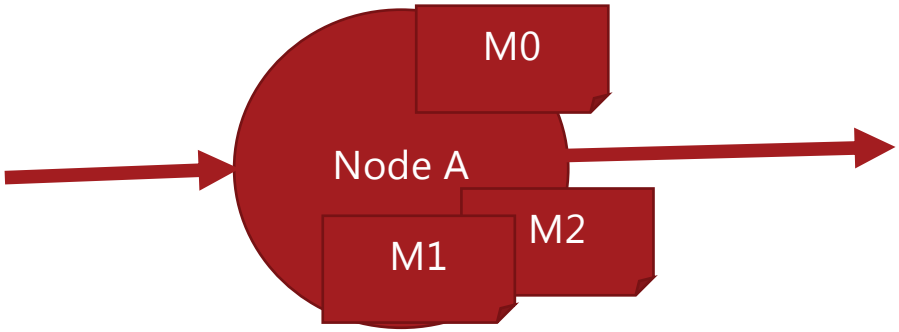
or

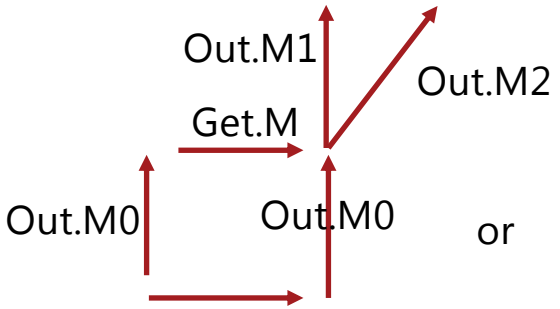


What is the input-output relation of the two versions of Node A?

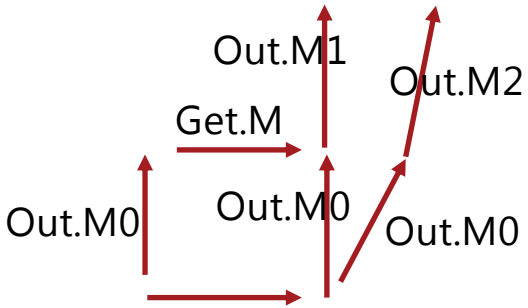
Non-deterministic Dataflow Network – input/output relations

After receiving a message and having output the message M0, then Node A may output either M1 or M2





or

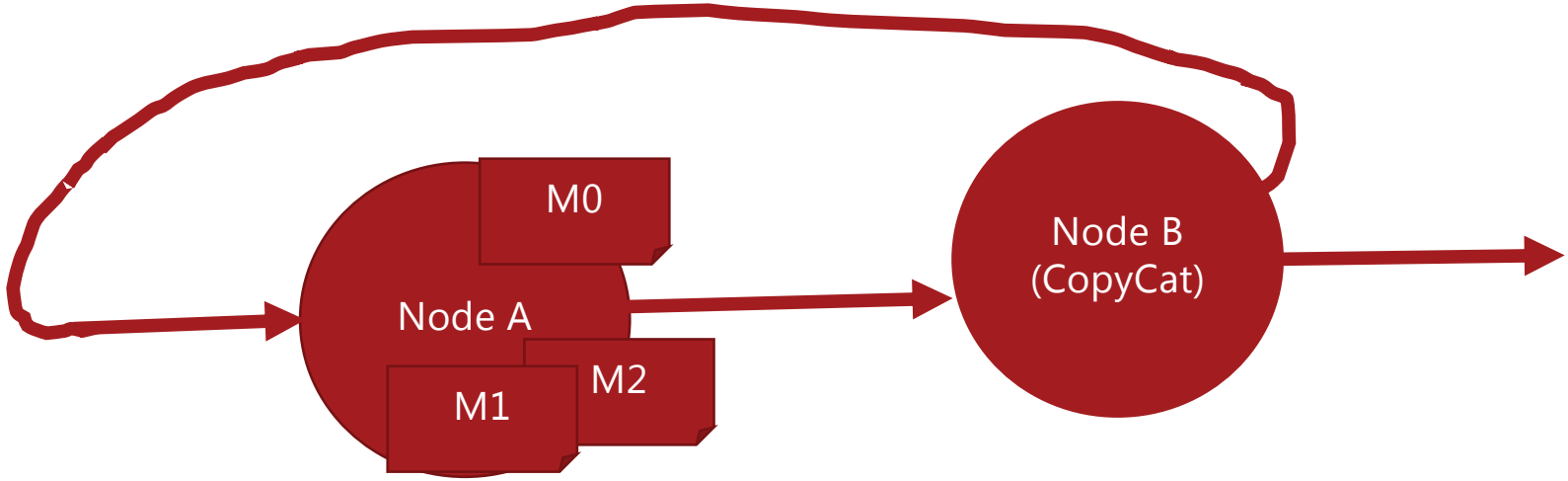


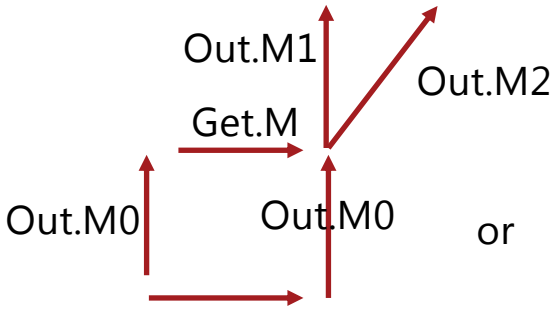
The two versions have the same Input/Output Relation

{(.,M0),
(M,M0),
(M,M0.M1),
(M,M0.M2)}

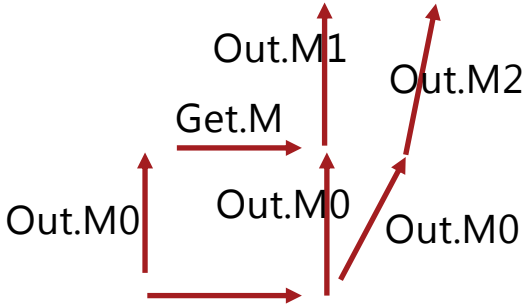
What is the input-output relation of the two versions of Node A?

Non-deterministic Dataflow Network – why I/O relations fail



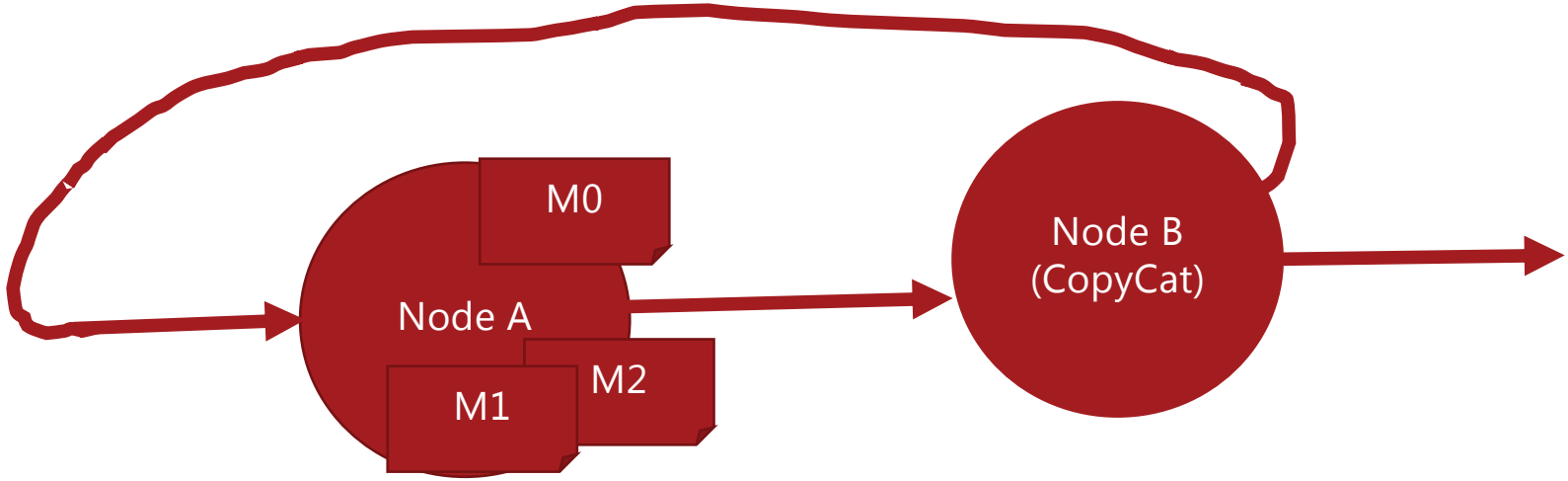


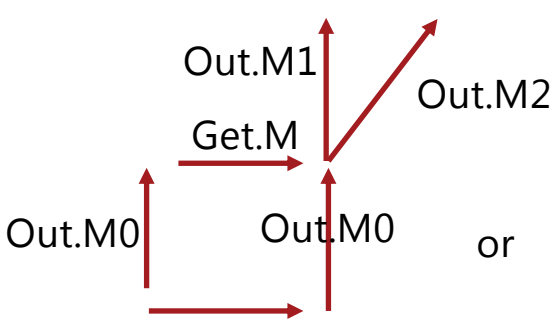
or



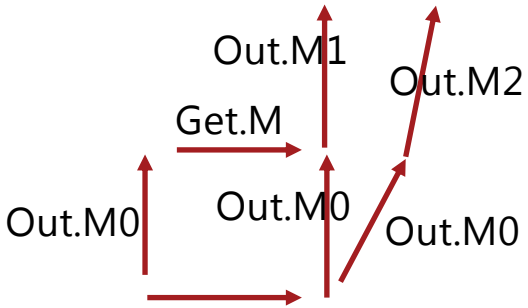
?

Non-deterministic Dataflow Network – why I/O relations fail





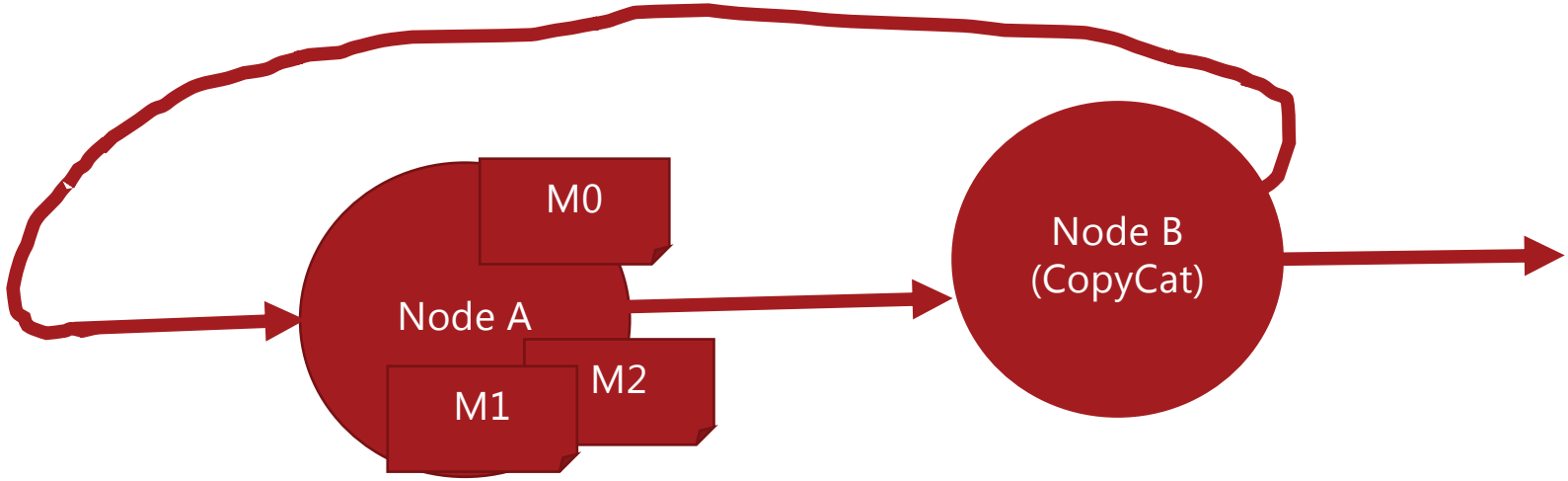
or

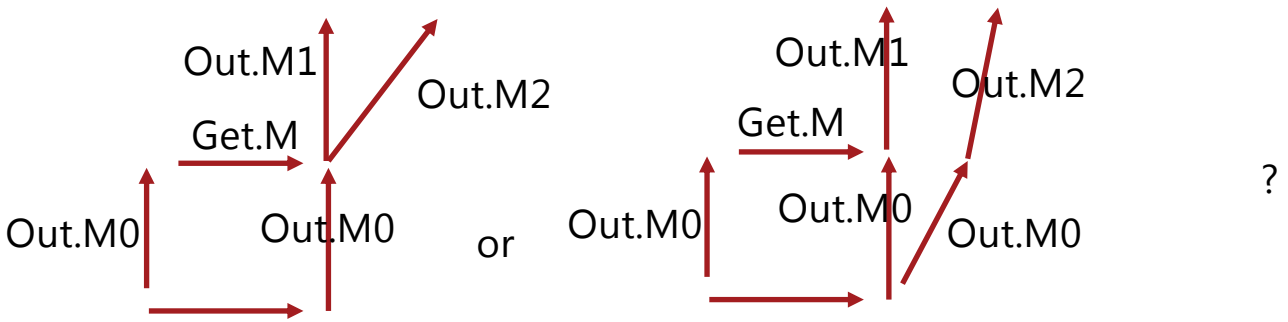


?

Does the Network behave the same for the two versions of Node A ?

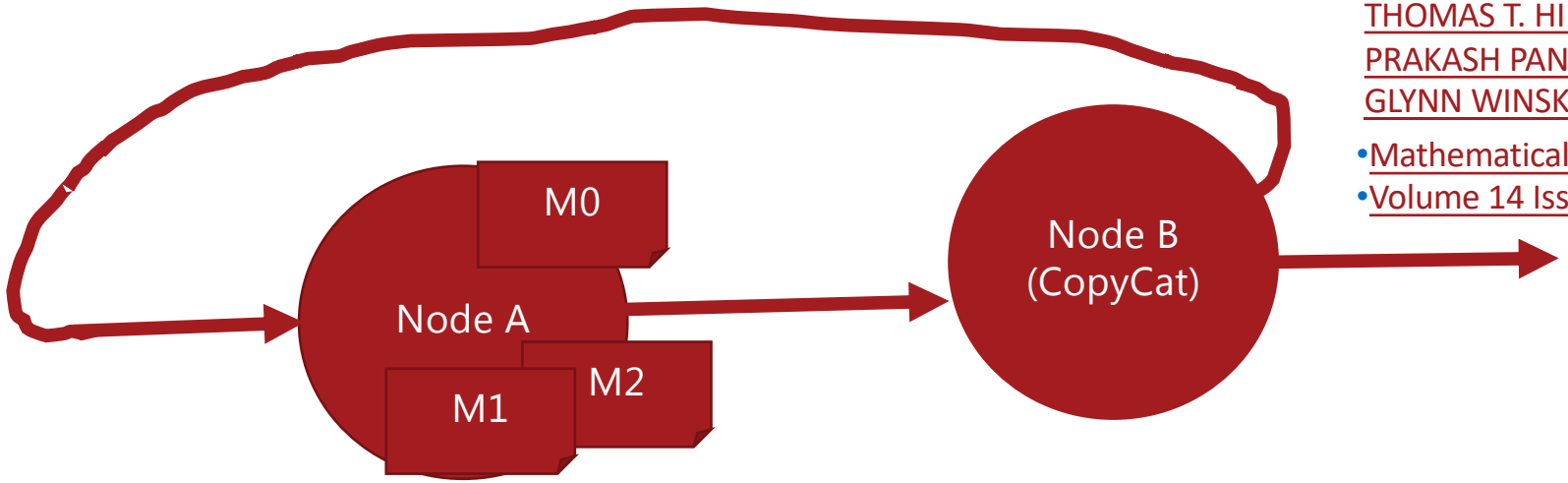
Non-deterministic Dataflow Network – why I/O relations fail





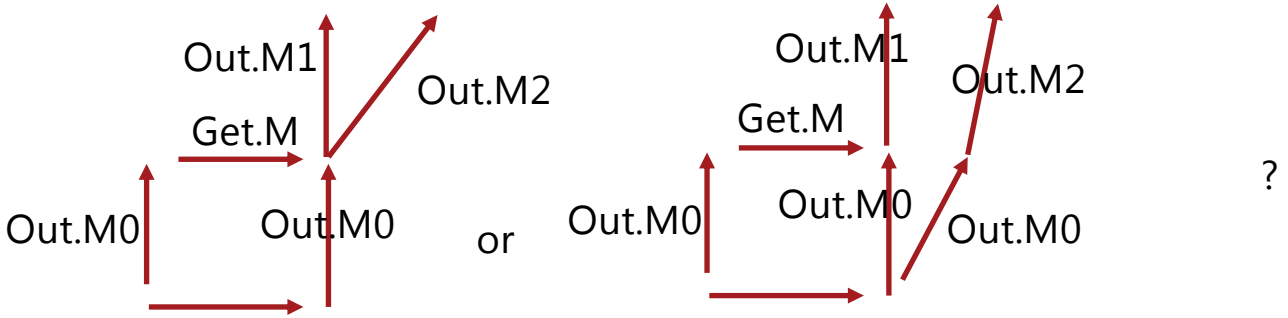
Does the Network behave the same for the two versions of Node A ? No – 2nd version only ouput M0.M1

Non-deterministic Dataflow Network



A relational model of non-deterministic dataflow
Published online by Cambridge University Press:
23 September 2004
THOMAS T. HILDEBRANDT,
PRAKASH PANANGADEN and
GLYNN WINSKEL

- Mathematical Structures in Computer Science
- Volume 14 Issue 5



So we need to capture the branching points in the semantics in order to give a compositional semantics!

Formal process languages for Distributed, reactive and event based systems ?

Process models & calculi

Lambda-calculus (30ties A. Church)

$$(\lambda x.\, t) \qquad (\lambda x.\, t)s \rightarrow t[x := s]$$

Actor Model (Hewitt ' 73)
Asynchronous messages (send, get) & spawning new actors

Dataflow Network ('74)

Programming languages & toolkits

Simula '67

ML '73
Miranda '85

Erlang '86

Haskell '90

Akka toolkit (2009)

Formal process languages for Distributed, reactive and event based systems ?

Process models & calculi

Lambda-calculus (30ties A. Church)

$$(\lambda x.\, t) \qquad (\lambda x.\, t)s \rightarrow t[x := s]$$

Actor Model (Hewitt ' 73)
Asynchronous messages (send, get) & spawning new actors

Dataflow Network ('74)

Communicating Sequential Processes (C. Hoare '80ties)

Calculus of communicating sequential processes (R. Milner '80)

$$P ::= 0 \mid a.P_1 \mid A \mid P_1 + P_2 \mid P_1|P_2 \mid P_1[b/a] \mid P_1 \backslash a$$

Programming languages & toolkits

Simula '67

ML '73
Miranda '85

Erlang '86

Haskell '90

Akka toolkit (2009)

Formal process languages for Distributed, reactive and event based systems ?

Process models & calculi

Lambda-calculus (30ties A. Church)

$$(\lambda x.\, t) \qquad (\lambda x.\, t)s \rightarrow t[x := s]$$

Actor Model (Hewitt ' 73)
Asynchronous messages (send, get) & spawning new actors

Dataflow Network ('74)

Communicating Sequential Processes (C. Hoare '80ties)

Calculus of communicating sequential processes (R. Milner '80)

$$P ::= 0 \mid a.P_1 \mid A \mid P_1 + P_2 \mid P_1|P_2 \mid P_1[b/a] \mid P_1 \backslash a$$

$\overline{a}.P_1 \mid a.P_2 \rightarrow P_1 \mid P_2$

Programming languages & toolkits

Simula '67

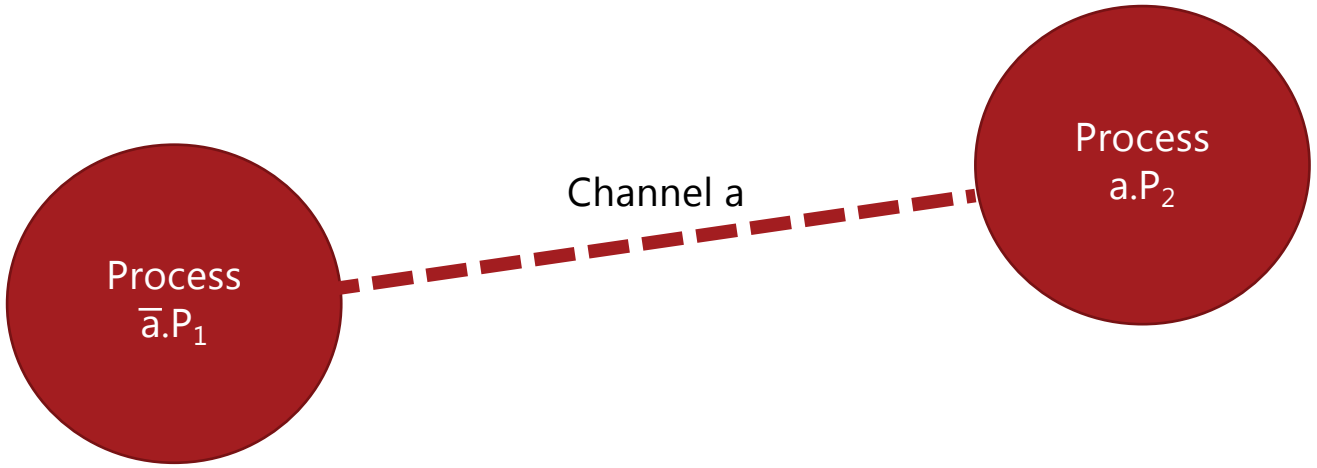
ML '73
Miranda '85

Erlang '86

Haskell '90

Akka toolkit (2009)

Process calculi: CCS



Messages exchanged synchronously $\overline{a}.P_1 \mid a.P_2 \rightarrow P_1 \mid P_2$

No shared state between processes

Non-determinism $\overline{a}.P_1 \mid (a.P_2 + a.P'_2) \rightarrow P_1 \mid P_2$ or $\overline{a}.P_1 \mid (a.P_2 + a.P'_2) \rightarrow P_1 \mid P'_2$

Fixed communication topology

From "An Introduction to Milner's CCS" (Luca Aceto, Kim G. Larsen & Anna Ingólfsdóttir)

What Are Reactive Systems ? A system that computes by reacting to stimuli from its environment

[6] D. HAREL AND A. PNUELI, *On the development of reactive systems*, in Logics and models of concurrent systems (La Colle-sur-Loup, 1984), vol. 13 of NATO Adv. Sci. Inst. Ser. F Comput. Systems Sci., Springer-Verlag, Berlin, 1985, pp. 477–498.

Example 1: "Coffee vending machine"

$$\mathrm{CM} \stackrel{\mathrm{def}}{=} \mathrm{coin}.\overline{\mathrm{coffee}}.\mathrm{CM} \quad . \qquad (1)$$

Example 2: "Coffee vending machine"

$$\mathrm{CTM} \stackrel{\mathrm{def}}{=} \mathrm{coin}.(\overline{\mathrm{coffee}}.\mathrm{CTM} + \overline{\mathrm{tea}}.\mathrm{CTM}) \quad . \qquad (2)$$

Example 3: "Computer Scientist produces a publication, but then needs a coffee before she can produce the next"

$$\mathrm{CS} \stackrel{\mathrm{def}}{=} \overline{\mathrm{pub}}.\overline{\mathrm{coin}}.\mathrm{coffee}.\mathrm{CS} \quad . \qquad (3)$$

Interface diagrams

$$\mathrm{CM} \stackrel{\mathrm{def}}{=} \mathrm{coin}.\overline{\mathrm{coffee}}.\mathrm{CM} \quad . \qquad (1)$$

$$\mathrm{CS} \stackrel{\mathrm{def}}{=} \overline{\mathrm{pub}}.\overline{\mathrm{coin}}.\mathrm{coffee}.\mathrm{CS} \quad . \qquad (3)$$

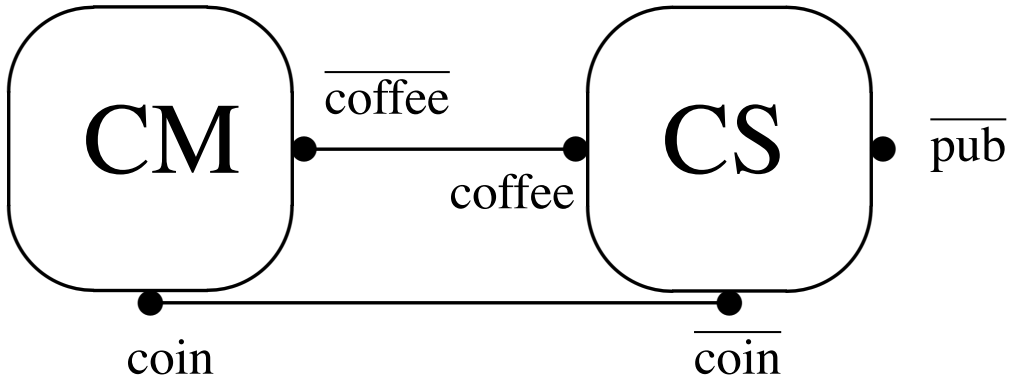


Table 2: The interface for process CM | CS

Transition systems

$$\mathrm{CS} \stackrel{\mathrm{def}}{=} \overline{\mathrm{pub}}.\overline{\mathrm{coin}}.\mathrm{coffee}.\mathrm{CS} \ .$$

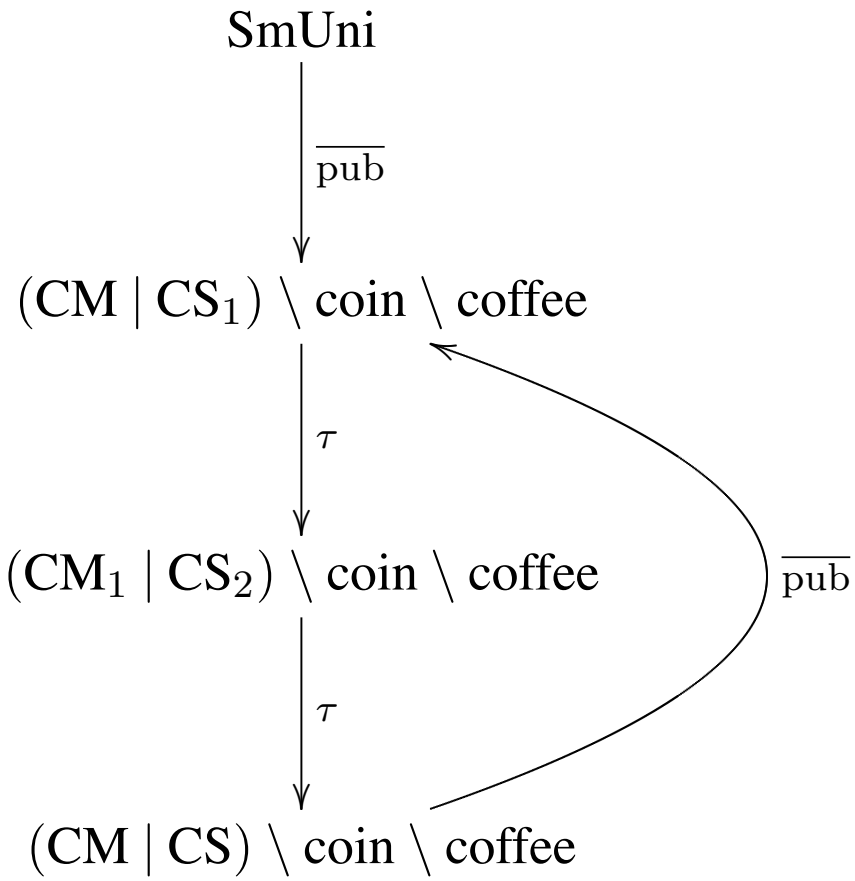
$$\mathrm{CS}_1 \stackrel{\mathrm{def}}{=} \overline{\mathrm{coin}}.\mathrm{coffee}.\mathrm{CS}$$

$$\mathrm{CS}_2 \stackrel{\mathrm{def}}{=} \mathrm{coffee}.\mathrm{CS} \ .$$

$$\mathrm{CS} \xrightarrow{\overline{\mathrm{pub}}} \mathrm{CS}_1 \xrightarrow{\overline{\mathrm{coin}}} \mathrm{CS}_2 \xrightarrow{\mathrm{coffee}} \mathrm{CS} \ .$$

Local communication ports

$$\mathrm{SmUni} \stackrel{\mathrm{def}}{=} (\mathrm{CM} \mid \mathrm{CS}) \setminus \mathrm{coin} \setminus \mathrm{coffee} \ .$$



Value passing CCS

$$\frac{}{\bar{a}(e).P \xrightarrow{\bar{a}(n)} P} \quad n \text{ is the result of evaluating } e$$

$$\frac{}{a(x).P \xrightarrow{a(n)} P[n/x]} \quad n \geq 0$$

$$\frac{P \xrightarrow{\alpha} P'}{\textbf{if bexp then } P \textbf{ else } Q \xrightarrow{\alpha} P'} \quad \text{bexp is true}$$

$$\frac{Q \xrightarrow{\alpha} Q'}{\textbf{if bexp then } P \textbf{ else } Q \xrightarrow{\alpha} Q'} \quad \text{bexp is false}$$

Formal process languages for Distributed, reactive and event based systems ?

Process models & calculi

Lambda-calculus (30ties A. Church)

$$(\lambda x.\, t) \qquad (\lambda x.\, t)s \rightarrow t[x := s]$$

Actor Model (Hewitt ' 73)
Asynchronous messages (send, get) & spawning new actors

Dataflow Network ('74)

Communicating Sequential Processes (C. Hoare '80ties)

Calculus of communicating sequential processes (R. Milner '80)

$$P ::= 0 \mid a.P_1 \mid A \mid P_1 + P_2 \mid P_1|P_2 \mid P_1[b/a] \mid P_1 \backslash a$$

$\overline{a}.P_1 \mid a.P_2 \rightarrow P_1 \mid P_2$

The π-calculus (pi-calculus) (Milner, Walker, Parrow '92)

$$\overline{x}\langle z\rangle.\, P|x(y).\, Q \rightarrow P|Q[z/y]$$

Programming languages & toolkits

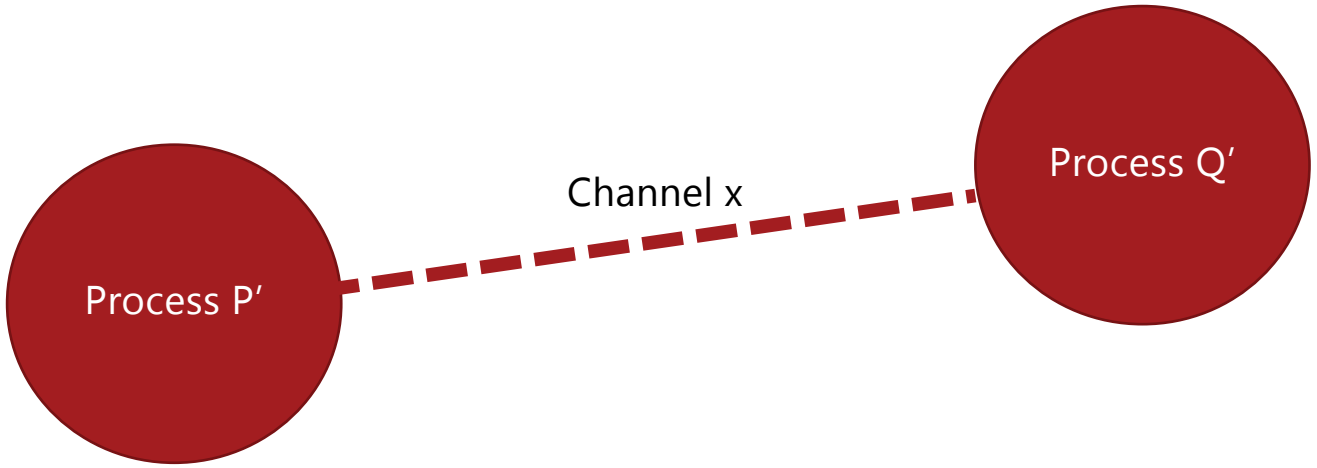
Simula '67

ML '73
Miranda '85

Erlang '86

Haskell '90

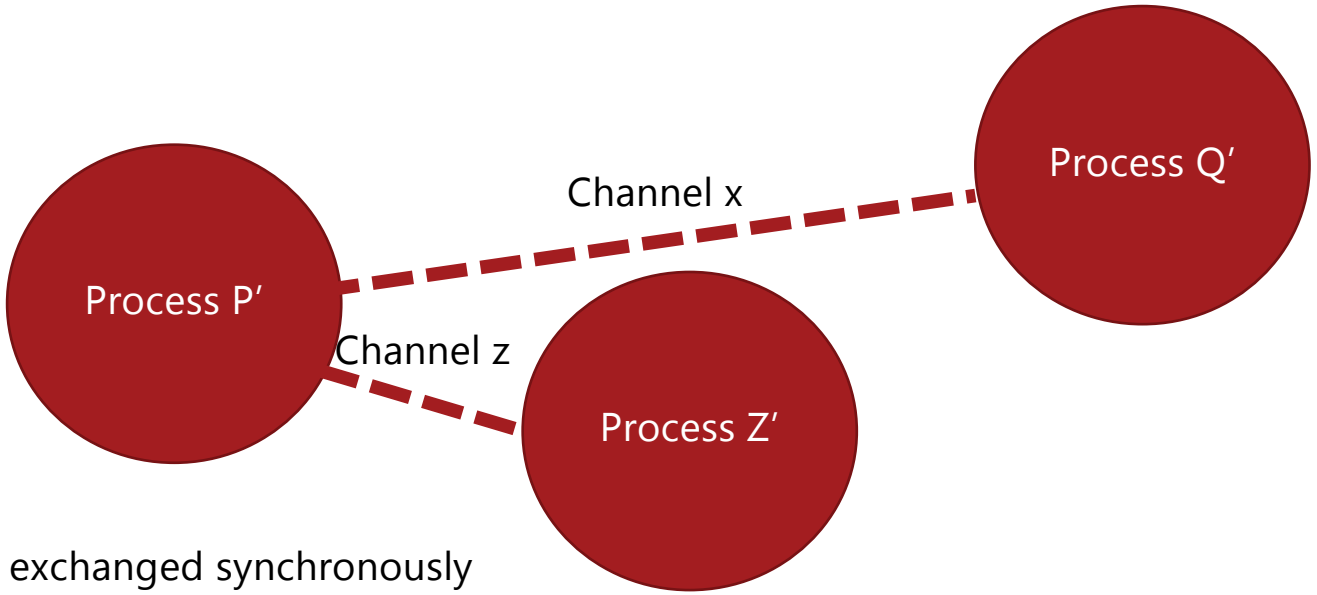
The π-calculus (pi-calculus) – Milner, Walker, Parrow



Messages exchanged synchronously
- Possibly passing names of channels!

$$\overline{x}\langle z\rangle.P|x(y).Q \rightarrow P|Q[z/y]$$

The π-calculus (pi-calculus) – Milner, Walker, Parrow



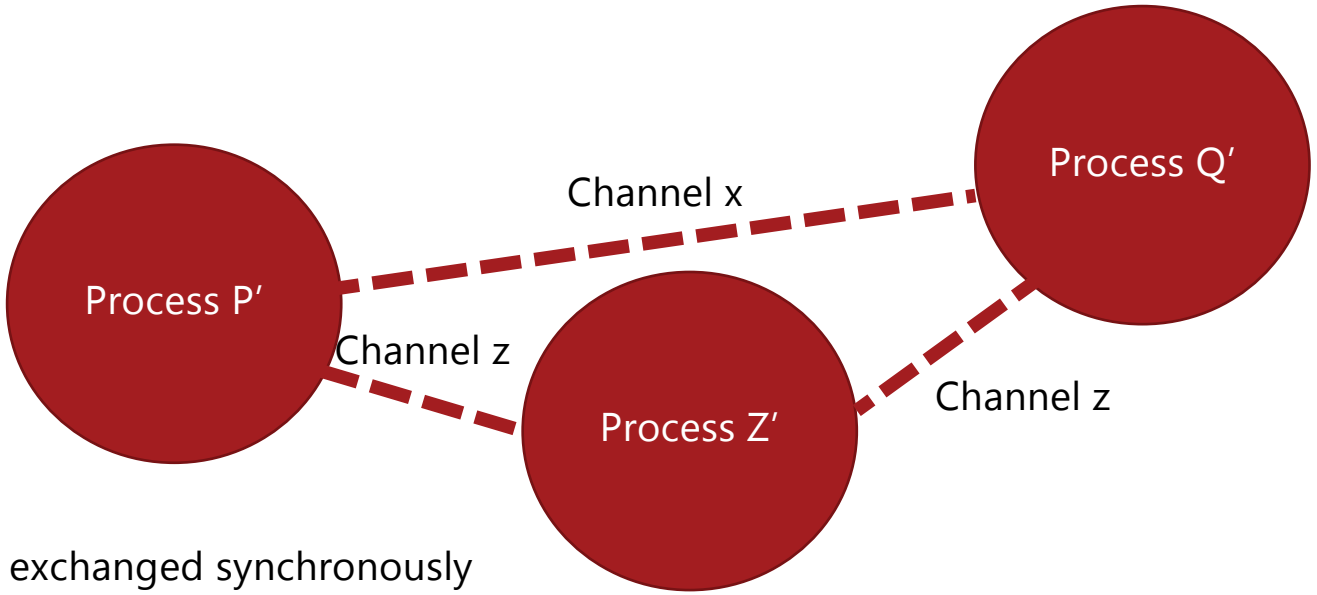
Messages exchanged synchronously
- Possibly passing names of channels!

$$\overline{x}\langle z\rangle.P|x(y).Q \rightarrow P|Q[z/y]$$

Changing communication topology!

$\overline{x}.\langle z\rangle.P \mid x(y).\overline{y}\langle a\rangle.Q \mid z(b).Z \rightarrow ?$

The π-calculus (pi-calculus) – Milner, Walker, Parrow



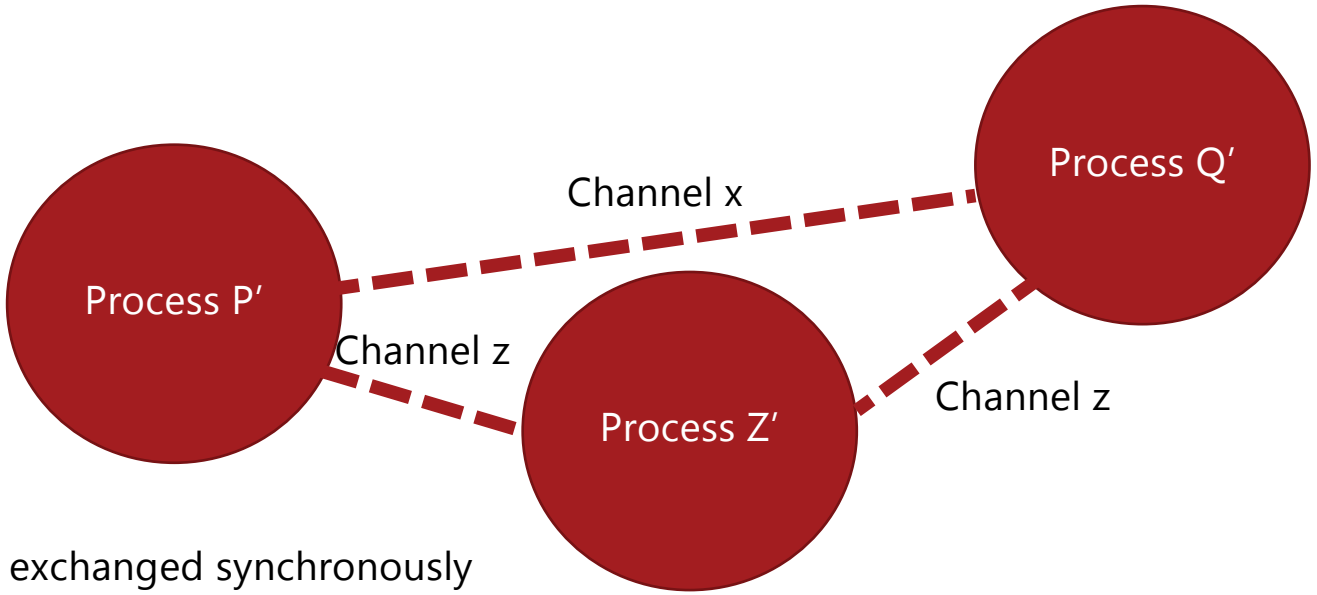
Messages exchanged synchronously
- Possibly passing names of channels!

Changing communication topology!

$$\overline{x}\langle z\rangle.P|x(y).Q \rightarrow P|Q[z/y]$$

$\overline{x}$.<z>.P | x(y).$\overline{y}$<a>.Q | z(b).Z → P | $\overline{z}$<a>.Q | Z

The π-calculus (pi-calculus) – Milner, Walker, Parrow



Messages exchanged synchronously
- Possibly passing names of channels!

$$\overline{x}\langle z\rangle.P|x(y).Q \rightarrow P|Q[z/y]$$

Changing communication topology!

$\overline{x}.\langle z\rangle.P \mid x(y).\overline{y}\langle a\rangle.Q \mid z(b).Z \rightarrow P \mid \overline{z}\langle a\rangle.Q \mid Z$

Still no shared state between processes

Still non-determinism

Full syntax of the π-calculus

$$
\begin{array}{rll}
P, Q ::= & x(y).\, P & \text{Receive on channel } x, \text{ bind the result to } y, \text{ then run } P \\
| & \overline{x}\langle y \rangle.\, P & \text{Send the value } y \text{ over channel } x, \text{ then run } P \\
| & P|Q & \text{Run } P \text{ and } Q \text{ simultaneously} \\
| & (\nu x)P & \text{Create a new channel } x \text{ and run } P \\
| & !P & \text{Repeatedly spawn copies of } P \\
| & 0 & \text{Terminate the process}
\end{array}
$$

Can encode lambda calculus.

Semantics given as labelled transition systems (similar to automata) and bisimulation equivalence.

More details in:

- Programming in the Pi-Calculus: A Tutorial Introduction to Pict, Benjamin C. Pierce, 1998
- A brief survey of the theory of the Pi-calculus. Daniel Hirschkof, 2003

Formal process languages for Distributed, reactive and event based systems ?

Process models & calculi

Lambda-calculus (30ties A. Church)

$$(\lambda x.\, t) \qquad (\lambda x.\, t)s \rightarrow t[x := s]$$

Actor Model (Hewitt ' 73)
Asynchronous messages (send, get) & spawning new actors

Dataflow Network ('74)

Communicating Sequential Processes (C. Hoare '80ties)

Calculus of communicating sequential processes (R. Milner '80)

$$P ::= 0 \mid a.P_1 \mid A \mid P_1 + P_2 \mid P_1|P_2 \mid P_1[b/a] \mid P_1 \backslash a$$

$\overline{a}.P_1 \mid a.P_2 \rightarrow P_1 \mid P_2$

The π-calculus (pi-calculus) (Milner, Walker, Parrow '92)

$$\overline{x}\langle z\rangle.\, P|x(y).\, Q \rightarrow P|Q[z/y]$$

Programming languages & toolkits

Simula '67

ML '73
Miranda '85

Erlang '86

Haskell '90

BPEL (2003)

Jolie (2007-)

Akka toolkit (2009)

The Jolie language (2007 -)

www.jolie-lang.org/

- Based on pi-like process calculus – but real language with C-like syntax

«The aim of Jolie is to offer native abstractions for the creation and composition of services»

```
type HelloRequest {
        name:string
}

interface HelloInterface {
requestResponse:
        hello( HelloRequest )( string )
}

service HelloService {
        execution: concurrent

        inputPort HelloService {
                location: "socket://localhost:9000"
                protocol: http { format = "json" }
                interfaces: HelloInterface
        }

        main {
                hello( request )( response ) {
                        response = "Hello " + request.name
                }
        }
}
```

** Service keyword and refinement types available from the upcoming 1.10 release.*

Review Questions

- What are the key primities in the Actor model ?
  - Advantages ? Challenges ? Examples of programming languages inspired by Actors ?
- What are the key primities of dataflow networks ?
  - What were the challenge giving semantics to non-deterministic data flow ?
- What are the key primitives of process calculi
  - CCS ?
  - Pi-calculus ?

Conclusions from lecture 4

- Development of formal models for concurrency and distributed systems and programming languages goes hand in hand
- Non-determinism gives rise to complex semantics and possibly surprising behaviour of even simple systems
- Actors and pi-like process calculi inspired programming languages and toolkits for message passing systems
- We will use the Jolie language for the rest of the course and next two assignments – version 1.10.5