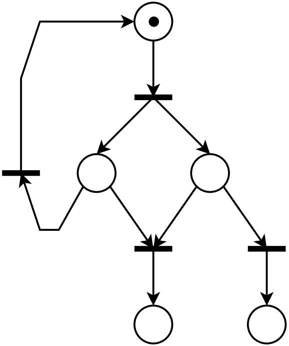
Process-oriented modelling

Ernesto Posse

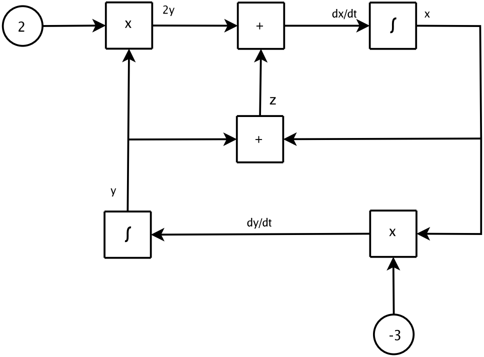
Modelling, Simulation and Design Lab
School of Computer Science
McGill University

September 25, 2006

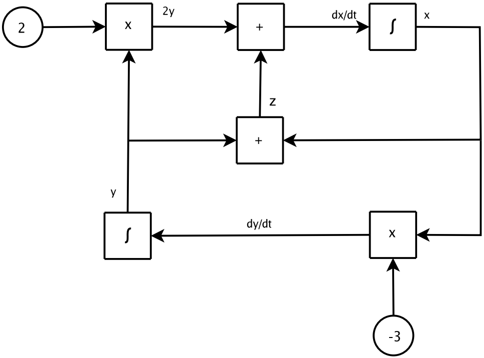
Petri Nets



Causal Block Diagrams



Causal Block Diagrams

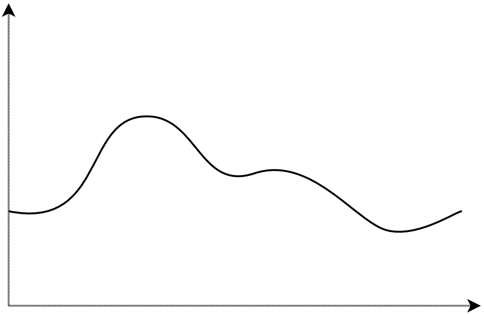


$$\begin{aligned}\frac{dx}{dt} &= 2y + z \\ \frac{dy}{dt} &= -3x \\ z &= x + y\end{aligned}$$

Causal Block Diagrams

- Variables are *signals*
- A *signal* is a function

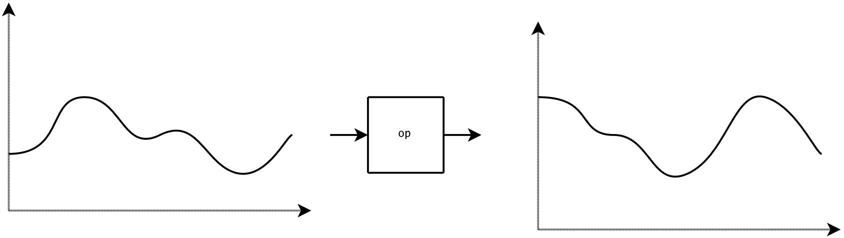
$$\mathbb{R} \longrightarrow \mathbb{R}$$



Causal Block Diagrams

- A *block* is a (higher-order) function:
- Unary block:

$$[\mathbb{R} \to \mathbb{R}] \to [\mathbb{R} \to \mathbb{R}]$$



Dataflow

- Discrete signals: time-base: $\mathbb{N}$, values: any set X

$$\mathbb{N} \to X$$

- This is the same as a *stream* of data:

$$\langle x_0, x_1, x_2, x_3, \ldots \rangle$$

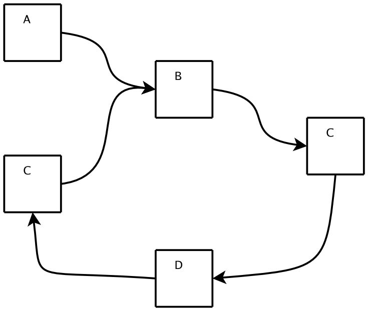
- Block: function

$$[\mathbb{N} \to X] \to [\mathbb{N} \to Y]$$

- A block takes one or more streams as input and produces a stream as output

Dataflow

- A *dataflow diagram* depicts a *process network*

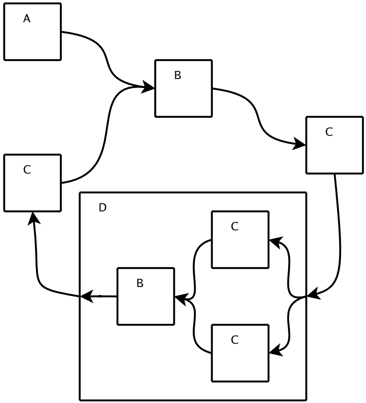


Dataflow

- Behaviour: transmission of information (tokens) as messages between processes (blocks) connected by channels.
- Concurrency: each process "runs" independently.
- Interactivity:
  - processes exchange information
  - the behaviour and output of an individual process depends on the input messages
  - A process doesn't need to terminate: continuous exchange of information

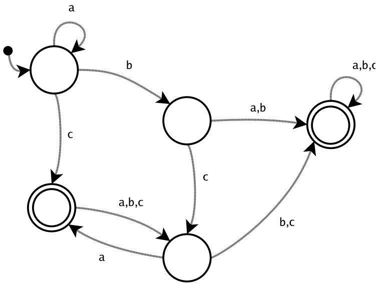
Dataflow

- Hierarchical composition (nesting): a process may be itself a process network



Dataflow

- Hierarchy base:
  - General dataflow: no particular model
  - Process-oriented view: state-machines

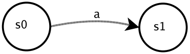


State-Transition systems

- Limitations of DFAs and NFAs:
  - Finite state-space (not so bad)
  - Finite alphabet (not so bad)
  - No notion of *rejection* (but can be easily emulated)
  - Determinism (not for NFAs; Whether this is a limitation depends on the problem.)
  - Termination: model of computation is, receive a full string (=stream) of input and finish producing one output (accept/reject)
  - No notion of interaction between automata

State-Transition systems

- Automata:
  - input: transition labels are *input* events



  - output: accept/reject states

State-Transition systems

- Transducers: a finite state machine that generates an output for each transition
  - Moore machines
  - Mealy machines

State-Transition systems

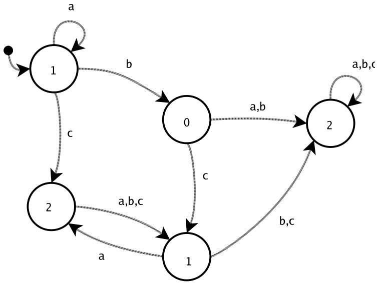
- Moore machines: transition labels are input events, states have outputs

$$(Q, q_0, X, Y, \delta, \lambda)$$

  - Q: A set of states
  - $q_0 \in Q$: a chosen initial state
  - X: an input alphabet
  - Y: an output alphabet
  - $\delta : Q \times X \to Q$: a transition function
  - $\lambda : Q \to Y$: an output function

State-Transition systems

- Moore machines: transition labels are input events, states have outputs



State-Transition systems

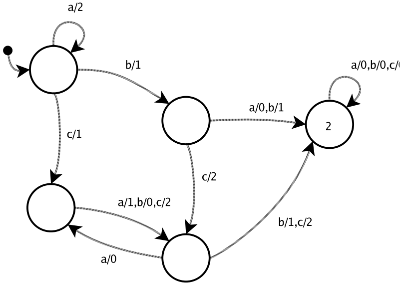
- Mealy machines: transition labels are input/output pairs

$$(Q, q_0, X, Y, \delta, \lambda)$$

  - Q: A set of states
  - $q_0 \in Q$: a chosen initial state
  - X: an input alphabet
  - Y: an output alphabet
  - $\delta : Q \times X \rightarrow Q$: a transition function
  - $\lambda : Q \times X \rightarrow Y$: an output function

State-Transition systems

- Mealy machines: transition labels are input events, states have outputs



State-Transition systems

- A transducer produces an output for every possible input
- Labelled Transition Systems (LTS)
  - No restriction on when output is produced
  - No restriction on finiteness of state-space or alphabet
  - Allows rejection (abscence of transitions)
  - Allows non-determinism

State-Transition systems

- An LTS is a tuple

$$(Q, A, \delta)$$

  - Q: set of states
  - A: set of labels
  - $\delta \subseteq Q \times A \times Q$: transition *relation*
  - Write

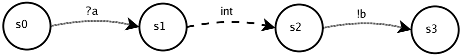
$$P \xrightarrow{a} P'$$

for

$$(P, a, P') \in \delta$$

State-Transition systems

- Labels may be interpreted as
  - input events
  - actions
- Actions:



  - observable:
    - input actions: ?a
    - output actions: !b
  - internal: int
- Process oriented view: labels are actions

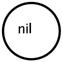
A language for state-transition systems

- Describing state-transition systems:
  - Diagrams
  - Mathematical notation
  - As a formal language
- Some formal languages used to describe *networks* of state transition systems:
  - CCS: Calculus of Communicating Systems
  - CSP: Concurrent/Communicating Sequential Processes

A language for state-transition systems

- Nil: a state that has no outgoing transitions (deadlock/termination)

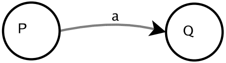
0



A language for state-transition systems

- Prefix (action)

$$P = a \to Q$$

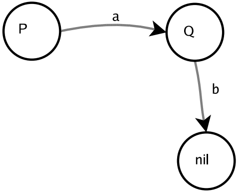


A language for state-transition systems

- Prefix (action)

$$P = a \longrightarrow Q$$

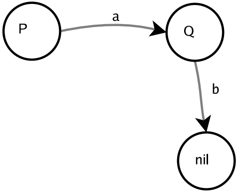
$$Q = b \longrightarrow 0$$



A language for state-transition systems

- Prefix (action)

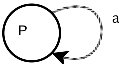
$$P = a \rightarrow b \rightarrow 0$$



A language for state-transition systems

- Loops: recursion

$$P = a \longrightarrow P$$

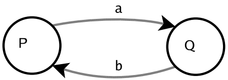


A language for state-transition systems

- Loops: recursion

$$P = a \rightarrow Q$$

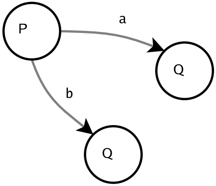
$$Q = b \rightarrow P$$



A language for state-transition systems

- Choice

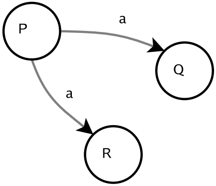
$$P = a \rightarrow Q + b \rightarrow R$$



A language for state-transition systems

- Choice: non-determinism

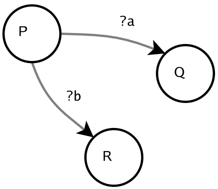
$$P = a \rightarrow Q + a \rightarrow R$$



A language for state-transition systems

- External choice: the environment decides

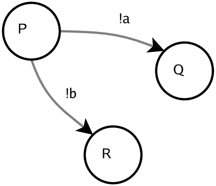
$$P = ?a \rightarrow Q + ?b \rightarrow R$$



A language for state-transition systems

- Internal choice: the system decides

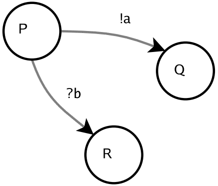
$$P = !a \rightarrow Q + !b \rightarrow R$$



A language for state-transition systems

- Internal choice: the system decides

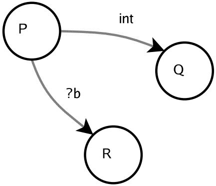
$$P = !a \rightarrow Q + ?b \rightarrow R$$



A language for state-transition systems

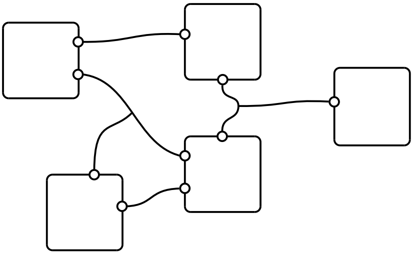
- Internal choice: the system decides

$$P = int \rightarrow Q + ?b \rightarrow R$$



A language for state-transition systems

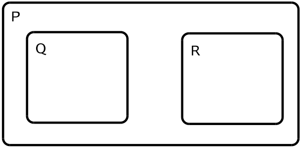
- Process networks



A language for state-transition systems

- Parallel composition

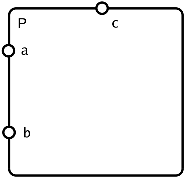
$$P = Q \parallel R$$



A language for state-transition systems

- Defining processes with ports

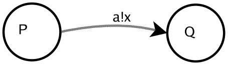
$$P(a, b, c) = ...a...b...c...$$



A language for state-transition systems

- Prefix (output actions)

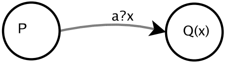
$$P(a) = a!x \rightarrow Q$$



A language for state-transition systems

- Prefix (input actions)

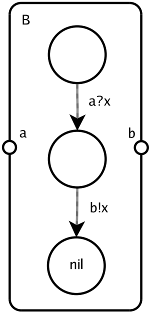
$$P(a) = a?x \rightarrow Q$$



A language for state-transition systems

- Example: data relay

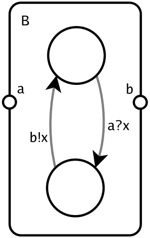
$$B(a, b) = a?x \longrightarrow b!x \longrightarrow 0$$



A language for state-transition systems

- Example: single cell-buffer

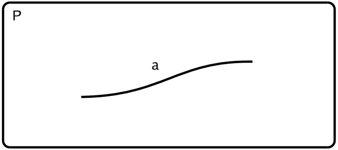
$$B(a,b) = a?x \rightarrow b!x \rightarrow B(a,b)$$



A language for state-transition systems

- Channels

$$\nu a.P$$



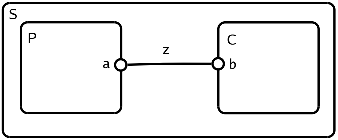
A language for state-transition systems

- Connecting processes

$$P(a) = ...a...$$

$$C(b) = ...b...$$

$$S = \nu z.(P(z) \parallel C(z))$$



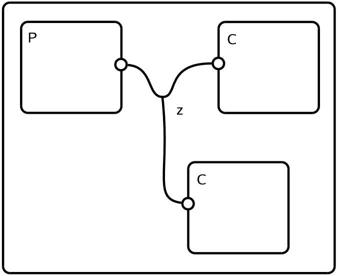
A language for state-transition systems

- Connecting processes

$$P(a) = \ldots a \ldots$$

$$C(b) = \ldots b \ldots$$

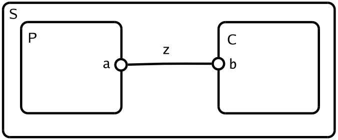
$$S = \nu z.(P(z) \parallel C(z) \parallel C(z))$$



A language for state-transition systems

- Communication: synchronous vs asynchronous

$$
\begin{aligned}
P(a) &= a!2 \rightarrow print.done \rightarrow 0 \\
C(b) &= b?x \rightarrow print.x \rightarrow 0 \\
S &= \nu z.(P(z) \parallel C(z))
\end{aligned}
$$



A language for state-transition systems

- Communication and nondeterminism

$$
\begin{aligned}
P(a) &= a!2 \rightarrow print.done \rightarrow 0 \\
C(b) &= b?x \rightarrow print.x \rightarrow 0 \\
S &= \nu z.(P(z) \parallel C(z) \parallel C(z))
\end{aligned}
$$