Introduction to Communicating Sequential Process (CSP) (Lecture 1)

Mannheim, September 2007

Contents

Administration

- Lecturer:
  - Leila Silva (leila@ufs.br)
  - Federal University of Sergipe, Brazil

Administration

- Classes:
  - Theory (11 classes)
  - Exercises and tools (5 classes)
  - Projects (6 classes)
  - Theroretical studies (2 classes)
  - Seminars (5 classes – project + theoretical studies)
  - Written examination (1 class)

Administration

- Examinations:
  - Written examination (last class) - 30%
  - Project – 30%
  - Report about theoretical study – 20%
  - Seminars – 20%
- Room: 1/212
- Laboratory: 1/010

Administration

References

1. C. A. R Hoare, *Communicating Sequential Processes*, Prentice Hall, 1985.
2. Steve Schneider, *Concurrent and Real-time Systems - The CSP Approach*, John Wiley & Sons, 2000.
3. A. W. Roscoe, *The Theory and Practice of Concurrency*, Pearson Education, 1997.
4. Classes notes of Dr. Jeff Sanders, Oxford University, 2004.
5. Classes notes of Dr. Alexandre Mota, UFPE, 2005.

Administration

- Tools
  - FDR and ProBE

    http://www.fsel.com/
  - JCSP

    http://www.cs.kent.ac.uk/projects/ofa/jcsp/

    http://www.ibm.com/developerworks/java/library/j-csp1.html
- Useful links

  http://www.comlab.ox.ac.uk/oucl/publications/books/concurrency

The Issues of Concurrency

- What is the study of concurrency?
  - Express and reason about systems of concurrent processes
  - Express: specify, design and implement
  - Reason: modify, compare, develop and verify
  - Concurrent systems: multiprocessors, operating systems, networks, reactive processes, etc...

The Issues of Concurrency

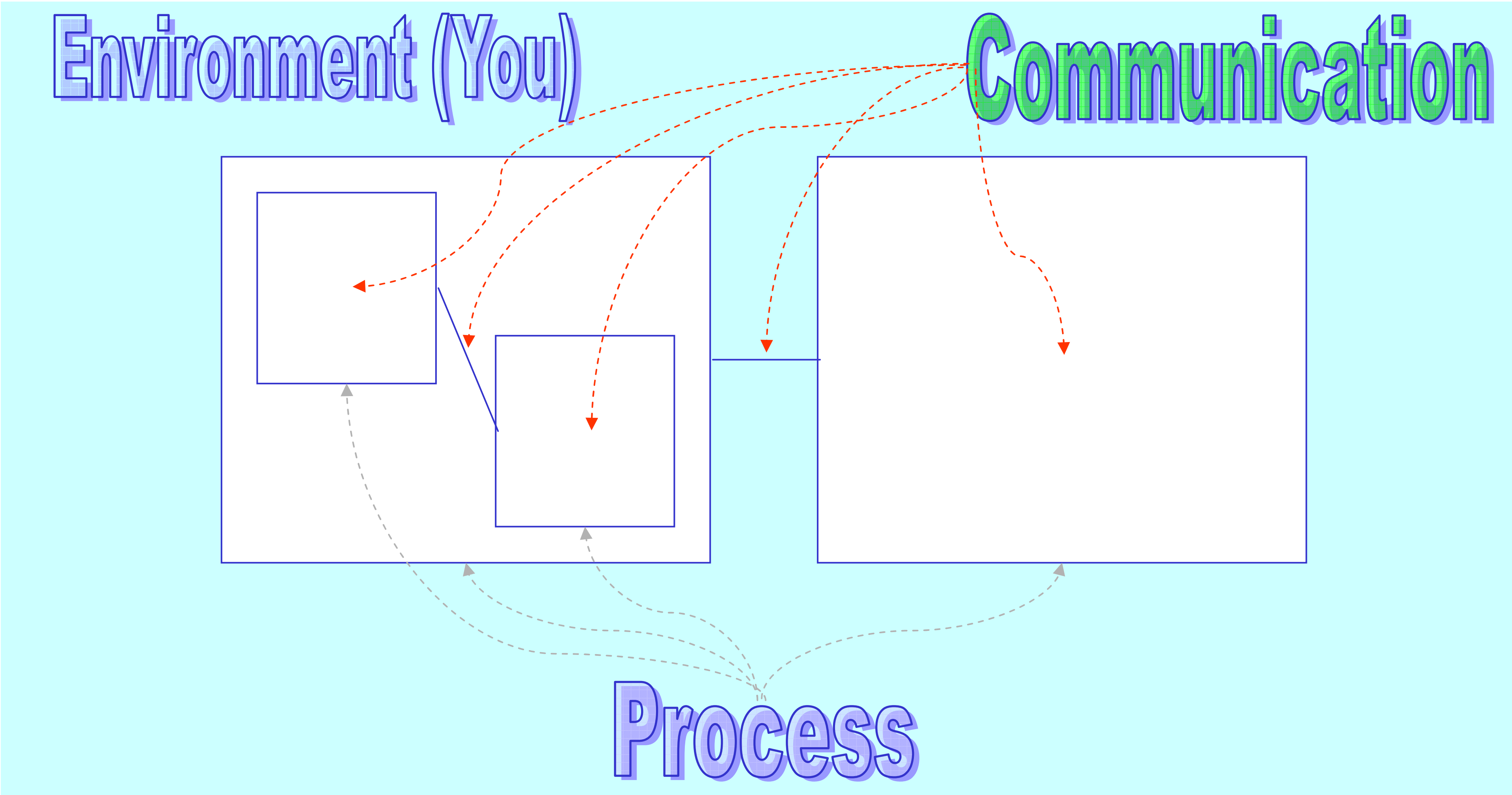
- The study of concurrence requires:
  - Formal notation with
    - large expressive power
    - laws for manipulating design
    - formal semantics
    - tools

CSP

CSP

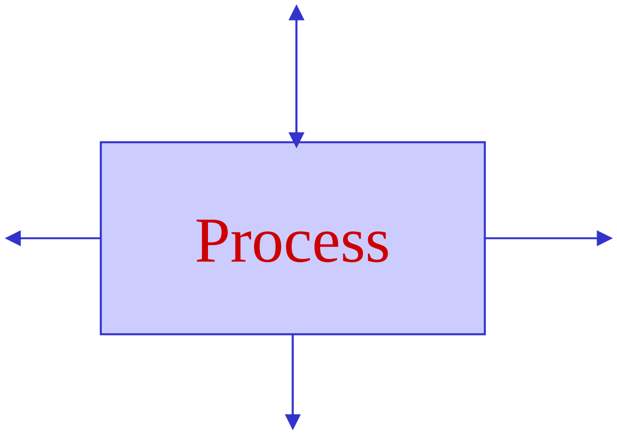
- Conceived to specify and reason about concurrent systems whose component processes interact with each other by communication.

Process



Process

- Independent entities, self-contained (black box), with particular interfaces used to interact with the environment (which is itself a process!)

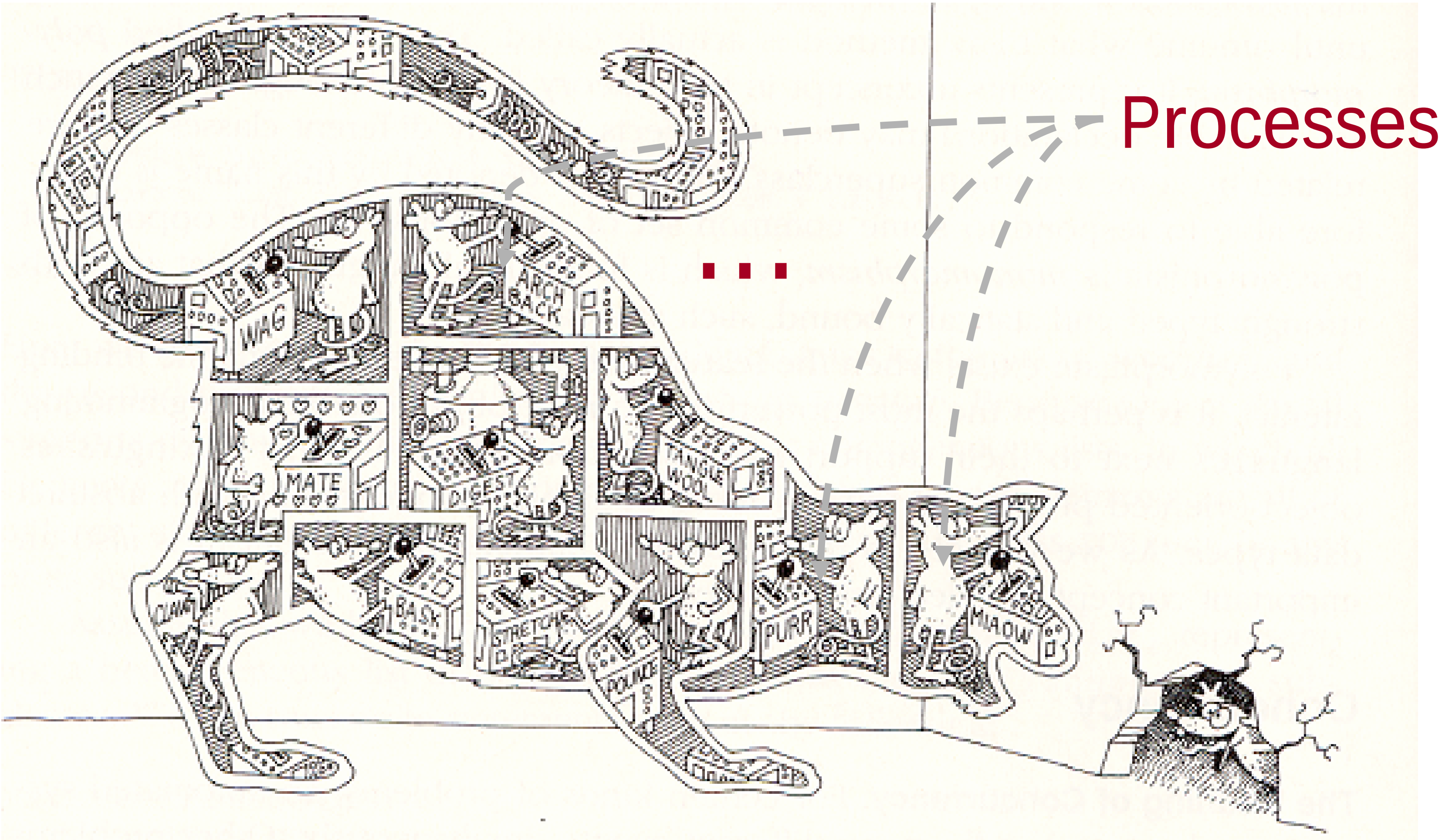


- In any run, a process performs a sequence of events.

Process

- Basic unit to capture behaviour.
- In general, we use a set of processes to get modularity.
- It is defined by equation(s)
  - P = (behaviour)
    - Similar to functional programming [although it is not a function!!!]
- Process names denote interesting system states/modules.

Process


Processes
...
WAG
ARCH BACK
MATE
DIGEST
TANGLE WOOL
BASK
STRETCH
POUNCE
PURR
MIAOW

Communication

- Term *communication* comes from the notion of interaction/observation/synchronisation
- It occurs between at least two parts [Which are?]
- A sequence of communications tells us a history (possible behaviour of a system → trace)
- A communication can be:
  - Event (no data communication → synchronisation)
  - Channel (a typed value is communicated)

Theory of Concurrency

- How is a theory of concurrency employed to implement a system?
  - Systems are decomposed into concurrently evolving processes.
  - Each process is treated as a system (and may be also decomposed into smaller systems)
- Any theory must handle:
  - the incremental approach: be compositional
  - change in level of abstraction: be hierarchical

Theory of Concurrency

- A main feature of concurrent systems is non determinism
  - Consequence of abstraction
  - A system is nondeterministic if it can exhibit different behaviors when given exactly the same inputs
  - An implementation is deterministic
  - In the development of a system, when passing to a less abstract description implies in removing nondeterminism
  - development => *refinement*

Theory of Concurrency

- Above nondeterminism any theory of concurrency must be able to model
  - Deadlock
    - No component can make any progress
  - Livelock
    - System performs infinite internal actions and does not communicate with the environment anymore

An overview of CSP

- CSP offers a succinct notation for processes and a way of controlling the level of abstraction.
- Provides basic operations
  - paralellism $||$
  - choice $\sqcap$, [], $\lhd$ $\rhd$
  - comunication ! ?
  - abstraction \

CSP

- Provides derived operations (*piping* (>>), *interleaving* (|||))
- Provides laws for those operations
- Semantics in layers ...traces...refusals...divergences...
- Tools

Processes

- What to specify? What is relevant to consider? How to do this?
- Processes interact among them through interfaces
- We need to abstract the internal actions and to focus on the interface: external actions
- Interaction / Communication
- The interface of a process is captured by a set of events

Events, Alphabets and Processes

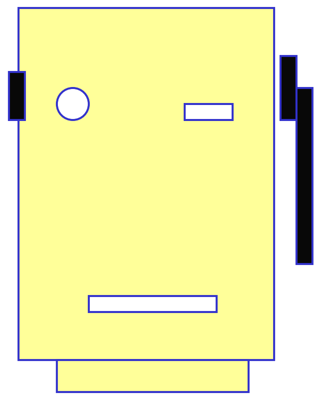
- Modelling requires a level of abstraction.
- When modelling a process we observe its *events*.
- Events are atomic and instataneous.
- A set of events forms the:
  - *alphabet* of the process being observed;
  - the *universe* under observation for several processes.
- *Alphabet*: αP
- *Universe* of a system: Σ

Processes

- A process exhibits a pattern of behaviour in which it offers certain events for synchronisation with its environment.
- Each event forms an interaction between the process and its environment.
- If the interaction does not occur the process blocks.

Processes

Example: Simple vending machine (*VMS*)



Events: *coin* and *choc*

Alphabet: $\alpha VMS = \{coin, choc\}$

Are there any alternatives?

Alphabets: Exercises

- A machine that accepts a 5p coins and returns the same amount in 1p and 2p coins.
- An alarm that emits beeps indefinitely.
- A board that only accepts up and down movements .

Alphabets: Exercises

- A machine that accepts a 5p coins and returns the same amount in 1p and 2p coins.

  $\alpha Chh: \{in5, out1, out2\}$

- An alarm that emits beeps indefinitely.

  $\alpha Alarm: \{beep\}$

- A board that only accepts up and right movements.

  $\alpha Board: \{up, right\}$

Processes

STOP

- Process that:
  - does not offer any event for communication;
  - performs no event, blocking them all.
- Represents a system (process)
  - in *deadlock,* or
  - Broken
- Terminal process

Processes

STOP

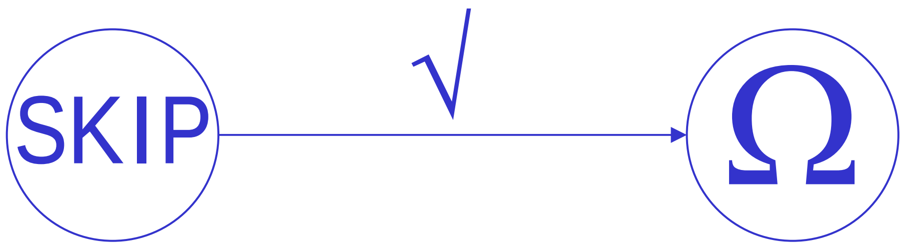
Processes

SKIP

- Process that does nothing and ends succesfully.
- Terminal process
- It only communicates a special event ($\surd$)
- After that, no communication and progress is possible

Processes

SKIP



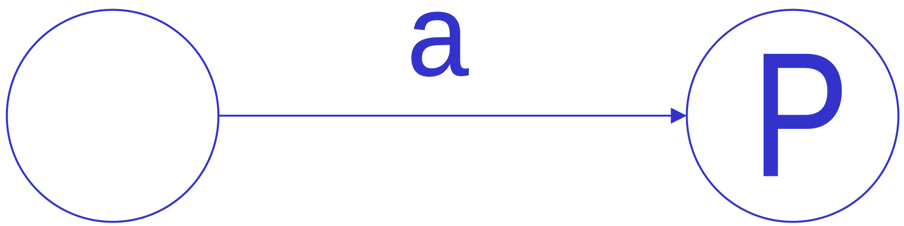
Prefix

- Most basic construct to model behaviour:

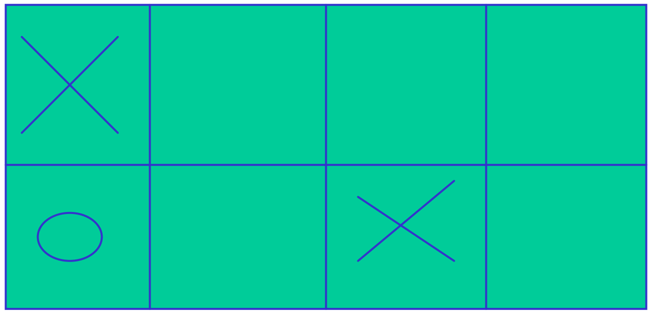
  a -> P (*a* is a *prefix* of *P*),

  given an event *a* and a process *P*.
- Offer an event *a* and wait, indefinitely, until a communication happens.
- After the communication, behaves like *P*.
- A process description that begins with a prefix is said to be *guarded*.

Prefix (a -> P)



Example



αBoard: {up, right}

Board: right -> up -> right -> right -> STOP

Processes: common errors

- For processes P and Q,

  $P \rightarrow Q$

  is wrong. Use sequential composition $P;$Q.
- For events a and b,

  $a \rightarrow b$

  is wrong. Use terminating process $STOP$ or $SKIP$ to write

  $a \rightarrow b \rightarrow STOP$ or $a \rightarrow b \rightarrow SKIP$

  depending on whether we wish to model unsuccessful or successful termination.

Processes: Exercises

1. A vending machine that accepts a coin and breaks.

2. A chocolate vending machine that serves two customers and breaks.

Processes: Exercises

1. A vending machine that accepts a coin and breaks.

$$\left(coin \rightarrow STOP\right)$$

2. A chocolate vending machine that serves two customers and breaks.

$$\left(coin \rightarrow (choc \rightarrow (coin \rightarrow (choc \rightarrow STOP)))\right)$$

Prefix: Precedence

- Excepting for the renaming operator, prefix has the highest precedence among CSP operators; it is right associative.

coin -> (choc -> STOP) = coin -> choc -> STOP

A Complete Specification

- It is a combination of
  - Alphabet declarations
  - Function and set definitions
    - *Odd(n) = (n % 2 == 1)*
    - *T = {0, 1, 2}*
  - And process definitions
    - *P = up -> down -> up -> down -> STOP*
- And, in general, following this ordering

Continuous Behaviour

- Various systems have a series of repetitive behaviour
- And these repetitions can be infinite
- Infinite behaviour becomes recursion
  - *P = (sequence of communications) -> P*
- Right-side process name is replaced by a left-side process definition

Recursion

Examples:

- A vending machine that ever function

  $V = coin \rightarrow choc \rightarrow coin \rightarrow choc \rightarrow \ldots$

  $V = coin \rightarrow choc \rightarrow V$

- A student who snores indefinitely:

  $\alpha Student = \{snore\}$

  $Student = snore \rightarrow Student$

Recursion Notation

- For guarded function F on processes, the solution to the recursive equation

  $$X = F\,X$$

  over alphabet A is written

  $$\mu X : A \bullet F\,X.$$

- Guarded means that $F\,X$ always starts with some event. E.g.

  $$F\,X = snore \rightarrow X.$$

Recursion Notation: Examples

- For a process X with alphabet $\{snore\}$ and $FX = snore \text{ -> } X$

  $$Student = \mu X: \underbrace{\{snore\}}_{A}.\ \underbrace{(snore\text{->}X)}_{FX}.$$

- For a process X with alphabet $\{coin, choc\}$ and $FX = coin \text{ -> } choc \text{ -> } X$

  $$VMS = \mu X: \underbrace{\{coin,choc\}}_{A}.\ \underbrace{(coin\text{->}choc\text{->}X)}_{FX}.$$

Recursion Notation: Exercises

1. Specify a clock that does anything but tick.

Recursion Notation: Exercises

1. Specify a clock that does anything but tick.

$\alpha CLOCK = \{tick\}$

$CLOCK = \mu X: \{tick\}. (tick\text{->}X)$

Equivalent to:

$CLOCK = (tick \rightarrow CLOCK)$

$= (tick \rightarrow (tick \rightarrow CLOCK))$ *by substitution*

$= (tick \rightarrow tick \rightarrow tick \rightarrow CLOCK)$ *by substitution*

$= \ldots = tick \rightarrow tick \rightarrow tick \rightarrow tick \ldots$

Recursion Notation: Exercises

2. A change machine which inputs *in5* and outputs change *out2* and *out1* in a fixed order.

Recursion Notation: Exerises

2. A change machine which inputs *in5* and outputs change *out2* and *out1* in a fixed order.

$\alpha Ch = \{in5, out1, out2\}$

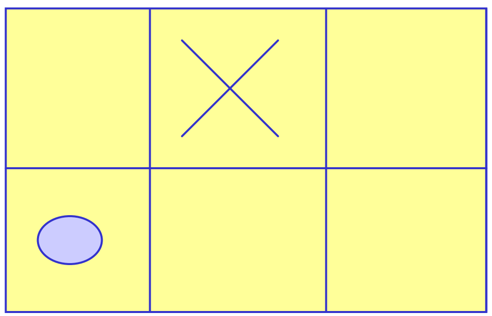
$Ch = in5 \rightarrow out2 \rightarrow out1 \rightarrow out1 \rightarrow out1 \rightarrow Ch$

$$Ch = \mu X : \{in5, out1, out2\} \bullet (in5 \rightarrow out2 \rightarrow out1 \rightarrow out1 \rightarrow out1 \rightarrow X).$$

Menu choice

- A process may offer its environment a choice of events.
- *Example:* $\alpha Board = \{up, right\}$

$$Board = (\; up \rightarrow STOP \;|\; right \rightarrow right \rightarrow up \rightarrow STOP)$$



Menu choice

- If a and b are distinct events, $a \neq b$, and P and Q are processes then

  $$(a \rightarrow P$$
  $$\mid b \rightarrow Q)$$

  is the process which offers its environment a choice of events a or b in which case it subsequently behaves like P or Q, respectively.

Menu choice

- Can be generalized to:

  $$(a_1 \;\text{->}\; P_1 \mid \ldots \mid a_n \;\text{->}\; P_n)$$

  given diferent events $a_1, \ldots, a_n$, and processes $P_1, \ldots, P_n$
- The choice symbol is not an operator between processes. Thus,

  $P \mid Q$ is wrong (syntax)

  $(x \to P) \mid (x \to Q)$ is wrong (syntax)

  $$(x \to P \mid y \to Q \mid z \to R) \neq (x \to P \mid (y \to Q \mid z \to R))$$

A single operator with three arguments

A single process

Wrong (syntax)

Menu choice: Example

- A more useful change machine offers the user a choice of changes

$\alpha Chh = \{in5, out1, out2\}$

$$\begin{aligned} Chh = in5 \rightarrow (&out2 \rightarrow out1 \rightarrow out1 \rightarrow out1 \rightarrow Chh \\ &\mid out1 \rightarrow out2 \rightarrow out2 \rightarrow Chh). \end{aligned}$$

Distinct events for a menu choice.

Menu choice: General Case Notation

The process which offers its environment a choice of an event in A

$$?x : A \longrightarrow P(x).$$

Scope of variable x .

- Cases:

$A = \{\}$: deadlock, $STOP$

$A = \{a\}$: prefix $a \text{ -> } P$

$A = \{a,b\}$: menu choice $(a \text{ -> } P(a) \mid b \text{ -> } P(b))$

$A = \{a_1,a_2,\ldots,a_n\}$: general choice $(a_1 \text{ -> } P(a_1) \mid a_2 \text{ -> } P(a_2) \mid \ldots \mid a_n \text{ -> } P(a_n))$

Menu Choice: Exercises

1. A machine that serves either chocolate or toffee on each transaction.

Menu Choice: Exercises

1. A machine that serves either chocolate or toffee on each transaction.

$$
\begin{aligned}
VM2 = \mu X : \{coin, choc, toffee\} \bullet \\
coin \rightarrow (choc \rightarrow X \mid toffee \rightarrow X)
\end{aligned}
$$

Menu Choice: Exercises

1. A machine that allows its customer to sample a chocolate and trusts him to pay after. The normal sequence of events is also allowed.

Menu Choice: Exercises

1. A machine that allows its customer to sample a chocolate and trusts him to pay after. The normal sequence of events is also allowed.

$$
\begin{aligned}
VM3 = \mu X : \{coin, choc\} \bullet (&coin \rightarrow choc \rightarrow X \\
\mid\ &choc \rightarrow coin \rightarrow X)
\end{aligned}
$$

Mutual Recursion

Several equations define the process behaviour

Example:

$$P_1 = up \rightarrow down \rightarrow P_1$$

can be defined by:

$$P_u = up \rightarrow P_d$$

$$P_d = down \rightarrow P_u$$

Mutual recursion

A rocket when in the air may be guided up or down; when on the ground it may be guided around.

$\alpha Rocket = \{up, down, around\}$

$Rocket = R_0 = (up \text{ -> } R_1 \mid around \text{ -> } Rocket)$

$R_{n+1} = (up \text{ -> } R_{n+2} \mid down \text{ -> } R_n)$

Mutual Recursion

$\alpha C = \{up, down, left, right\}$

$C = (up \text{ -> } D \mid left \text{ ->} A \mid right \text{ -> } E)$

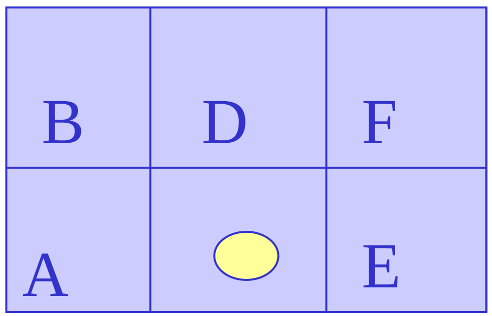
$D = (down \text{ -> } C \mid left \text{ -> } B \mid right \text{ -> } F)$

$E = (up \text{ -> } F \mid left \text{ -> } C)$

$F = (down \text{ -> } E \mid left \text{ -> } D)$

$A = (up \text{ -> } B \mid right \text{ -> } C)$

$B = (down \text{ -> } A \mid right \text{ -> } D)$



Mutual Recursion: Exercises

1. The process *LIGHT* may be defined in terms of process *ON*, which is itself defined in terms of process *LIGHT*. The events are *on* and *off*.

Mutual Recursion: Exercises

1. The process *LIGHT* may be defined in terms of process *ON*, which is itself defined in terms of process *LIGHT*. The events are *on* and *off*.

$$LIGHT = on \rightarrow ON$$

$$ON = off \rightarrow LIGHT$$

Mutual Recursion: Exercises

2. A counter can be incremented (*up*) or decremented (*down*) at any point, provided the total number of decremented events does not exceed the number of increment events.

Mutual Recursion: Exercises

2. A counter can be incremented (*up*) or decremented (*down*) at any point, provided the total number of decremented events does not exceed the number of increment events.

$$COUNT_0 = up \rightarrow COUNT_1$$

$$COUNT_n = (\ up \rightarrow COUNT_{n+1}$$
$$\qquad\qquad | \ down \rightarrow COUNT_{n-1}) \quad (n > 0)$$

Communication Events

- The event consisting of communication of value *v* on channel *c* is written *c.v*. Usually a communication event results from an input and output ocurring in parallel.
- Output event: *c ! e*
- Input event: *c ? x*
- Example:

  Copy = in ? x -> out ! x -> Copy

Communication Events

- The set of all events communicated allong channel c is denoted $\{|c|\}$.
- Example:
  - If process *Copy* has channel *in* of integer type then

    $$\{|in|\} = \{in.z \mid z \in Z\}$$

Communication Events

An accumulator is used to keep track of running totals of sequences of numbers. It has a *reset* event, a *query* channel on which the current total can be output, and an *add* channel where it is possible to add another number.

$$Accumulator = R_0$$

$$R_i = (reset \text{ -> } R_0 \mid query\,!\,i \text{ -> } R_i \mid add?x{:}N \text{ -> } R_{i+x})$$

Events vs Channels

- They are conceptually distinct
- But, in practice, a channel is indeed a set of events
- Thus, the channel a:{0,1} is the set of events {a.0, a.1}
  - Note the use of the . operator as a separator
  - Channels simply have a more elegant and readable presentation than events

Communication Events: Exercises

1. A process that duplicates its input data.

2. A process that given two natural numbers, outputs their product.

Communication Events: Exercises

1. A process that duplicates its input data.

Dup = in ?x -> out ! x -> out ! x -> Dup

2. A process that given two natural numbers, outputs their product.

*Prod = in ? x:N -> in ? y:N -> out ! (x*y) -> STOP*