Cyber-Physical Systems (CSC.T431)

Asynchronous Model (1)

Instructor: Takuo Watanabe (Department of Computer Science)

Agenda

- Asynchronous Model (1)

Course Support & Material

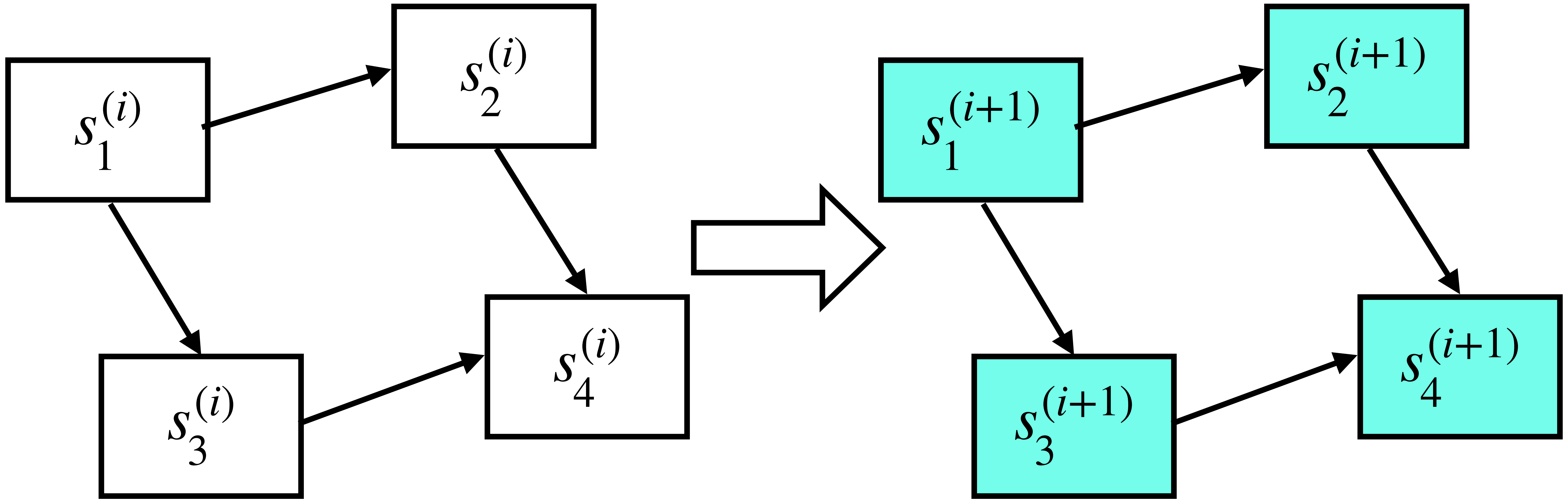
- Slides: OCW-i
- Course Web: https://titech-cps.github.io
- Course Slack: titech-cps.slack.com

Asynchronous Model

Asynchronous Processes

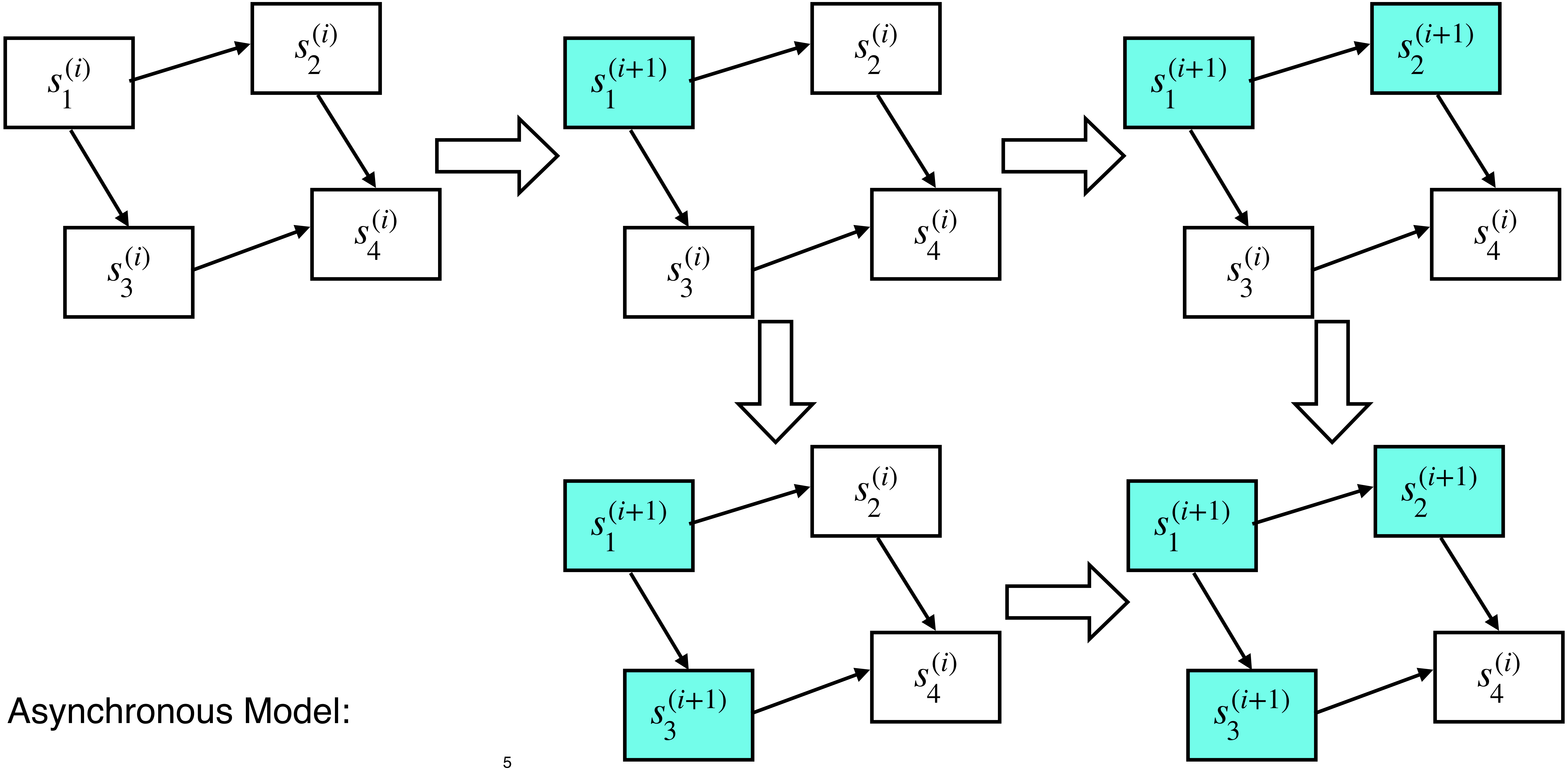
- An *asynchronous process* interacts with other asynchrinous processes via *input/output channels* and maintains its internal state
- The execution speeds of different processes are independent.
  - The reception of inputs is decoupled from the production of outputs.
  - Any internal computation takes an unknown but nonzero amount of time.

Synchronous Model



Lock-Step Execution

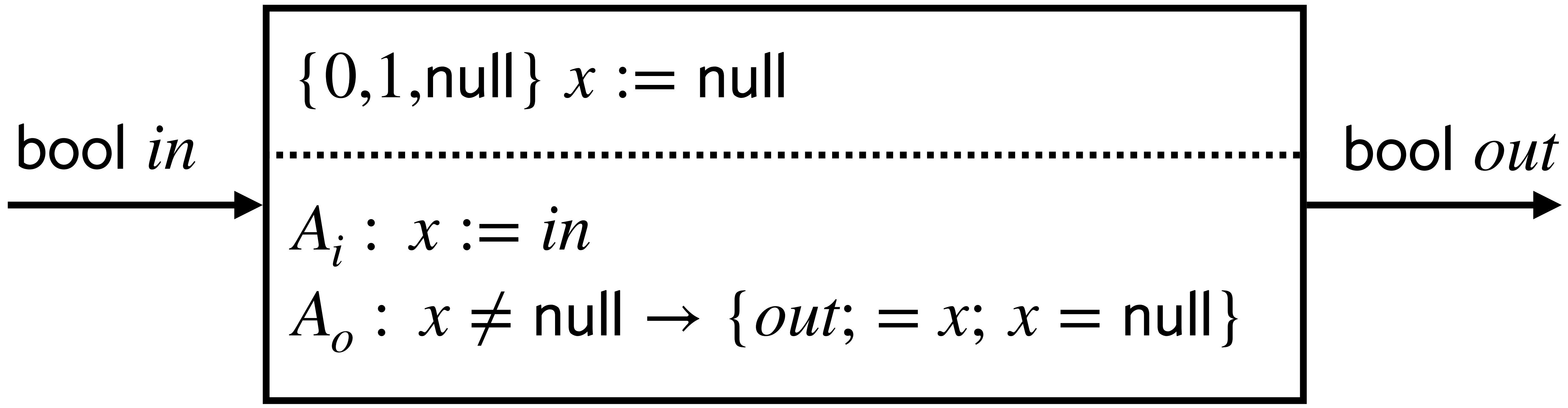
Asynchronous Model



Asynchronous Model:

Asynchronous Process

Ex. Buffer



- in : Boolean input channel
- out : Boolean output channel
- x : state variable of type $\{0,1,\text{null}\}$ $(= \text{bool} \cup \{\text{null}\})$, initialized as null.
- A_i : input task
- A_o : output task

Input and Output Channels

- An asynchronous process has a set of typed *input channels* and a set of typed *output channels*.
- $x?v$: We write $x?v$ to denote an *input* from an input channel x, where v is a value of the type of x. It can also be interpreted as *receiving* v *on* x.
- $y!v$: We write $y!v$ to denote an *output* to an output channel y, where v is a value of the type of y. It can also be interpreted as *sending* v *on* y.
- In each computational step (corresponds to a round in synch. model), a process can handle at most one input or output, even when the process has multiple input/output channels.

Input Tasks

- Processing of an input is called an *input action*, written as $s \xrightarrow{x?v} t$.
  - In an input action, the process can only update its state and does not produce outputs.
- An input action is specified by an input task.
  - Each input task is associated with a single input channel.
  - $\mathbf{A}_x$ denotes the set of input tasks associated with the input channel x.
- An input task $A \in \mathbf{A}_x$ can be described as $Guard \rightarrow Update$ where:
  - $Guard$: a condition on the states, defining $[\![Guard]\!] \subseteq Q_S$.
  - $Update$: description of the updates of the states. If $s \in [\![Guard]\!]$ and $x?v$ is an input, then $(s[x \mapsto v], t) \in [\![Update]\!]$ for some $t \in Q_S$. This corresponds to $s \xrightarrow{x?v} t$.

Output Tasks

- Producing an output is called an *output action*, written as $s \xrightarrow{x!v} t$.
  - In an output action, the process can only update its state and does not process inputs.
- An output action is specified by an *output task*.
  - Each output task is associated with a single output channel.
  - $\mathbf{A}_y$ denotes the set of output tasks associated with the output channel y.
- An output task $A \in \mathbf{A}_y$ can be described as $Guard \rightarrow Update$ where:
  - $Guard$: a condition on the states, defining $[\![Guard]\!] \subseteq Q_S$.
  - $Update$: description of the updates of the states. If $s \in [\![Guard]\!]$ and $x!v$ is an output, then $(s, t[y \mapsto v]) \in [\![Update]\!]$ for some $t \in Q_S$. This corresponds to $s \xrightarrow{x!v} t$.

Ex. Buffer

- Input Channels: $\{in\}$, defining the set of inputs $\{in?0, in?1\}$.
- Output Channels: $\{out\}$, defining the set of outputs $\{out!0, out!1\}$.
- State Variables: $\{x\}$, defining the set of states $Q_S = \{0, 1, \text{null}\}$.
- Initialization: $\{0,1,\text{null}\}\ x := \text{null}$, defining the set of initial states $\{\text{null}\}$.
- Input Tasks: $\mathbf{A}_{in} = \{A_i\}$, where $A_i : 1 \rightarrow x := in$, defining the set of input actions $\{s \xrightarrow{in?v} v \mid s \in Q_S\}$.
- Output Tasks: $\mathbf{A}_{out} = \{A_o\}$, where $A_o : x \neq \text{null} \rightarrow \{out := x;\ x := \text{null}\}$, defining the set of output actions $\{v \xrightarrow{out!v} \text{null} \mid v \neq \text{null}\}$.

Internal Tasks

Ex. AsyncInc

nat $x := 0; y := 0$

$A_x : \; x := x + 1$
$A_y : \; y := y + 1$

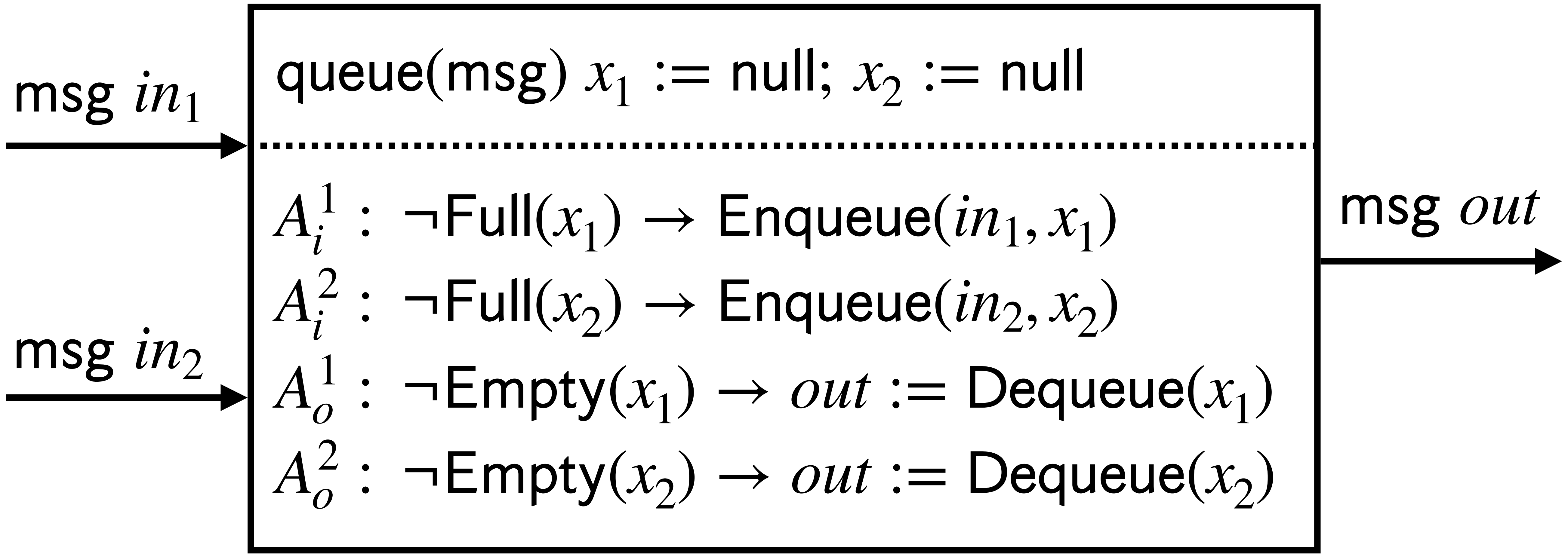
- The process AsyncInc does not have input/output channels.
- Tasks that neither process inputs nor produce outputs are called *internal tasks*.
  - A_x and A_y in AsyncInc are internal tasks.
- Internal tasks describe *internal actions*, written as $s \xrightarrow{\varepsilon} t$.
  - $A_x : x := x + 1$ defines the set of internal actions $\{(i, j) \xrightarrow{\varepsilon} (i + 1, j) \mid i, j \in \mathbb{N}\}$.
  - $A_y : y := y + 1$ defines the set of internal actions $\{(i, j) \xrightarrow{\varepsilon} (i, j + 1) \mid i, j \in \mathbb{N}\}$.
  - The set of internal actions is $\{(i, j) \xrightarrow{\varepsilon} (i + 1, j) \mid i, j \in \mathbb{N}\} \cup \{(i, j) \xrightarrow{\varepsilon} (i, j + 1) \mid i, j \in \mathbb{N}\}$.

Enabled Tasks

- Let $A : Guard \to Update$ is a task (input, output, or internal). If a state s satisfies $Guard$, A is said to be *enabled* in s.

- Note again that $Guard$ is a condition over the states of the process. This means that it should not refer to input/output channels.
- If A is an input task, $Update$ may read at most one input channel.
- If A is an output task, $Update$ may update at most one output channel.
- If A is an internal task, $Update$ may only update state variables.

Ex. Merge



$$\mathbf{A}_{in_1} = \{A_i^1\}$$
$$\mathbf{A}_{in_2} = \{A_i^2\}$$
$$\mathbf{A}_{out} = \{A_o^1, A_o^2\}$$

- If $\mathrm{Full}(x_1) \wedge \mathrm{Full}(x_2)$, no input tasks can be executed until an output action happens.
- If $\mathrm{Empty}(x_1) \wedge \mathrm{Empty}(x_2)$, no output tasks can be executed until an input action happens.
- If $\neg\mathrm{Full}(x_1) \wedge \neg\mathrm{Full}(x_1)$, one of A_i^1 and A_i^2 is nondeterministically chosen and executed.
- If $\neg\mathrm{Empty}(x_1) \wedge \neg\mathrm{Empty}(x_1)$, one of A_o^1 and A_o^2 is nondeterministically chosen and executed.

Asynchronous Process

Formal Definition

- An asynchronous process $P = (I, O, S, Init, \mathcal{A}_I, \mathcal{A}_O, \mathbf{A})$ consists of:
  - I : a finite set of typed *input channels*
  - O : a finite set of typed *output channels*
  - S : a finite set of typed state variables
  - $Init$: a description of the *initialization* defining the set $[\![Init]\!] \subseteq Q_S$ of initial states
  - $\mathcal{A}_I = \{\mathbf{A}_x \mid x \in I\}$ where $\mathbf{A}_x$ is the set of *input tasks* for input channel x
  - $\mathcal{A}_O = \{\mathbf{A}_y \mid y \in O\}$ where $\mathbf{A}_y$ is the set of *output tasks* for output channel y
  - $\mathbf{A}$: set of *internal tasks*

Tasks

- Input Tasks: $\mathscr{A}_I = \{A_x \mid x \in I\}$
  - An input task $A \in A_x$ can be described as $Guard \to Update$, where $[\![Guard]\!] \subseteq Q_S$ and $[\![Update]\!] \subseteq Q_{S \cup \{x\}} \times Q_S$, defining the set of input actions $\{s \xrightarrow{x?v} t \mid s \in [\![Guard]\!] \wedge (s[x \mapsto v], t) \in [\![Update]\!]\}$.
- Output Tasks: $\mathscr{A}_O = \{\mathbf{A}_y \mid y \in O\}$
  - An output task $A \in \mathbf{A}_y$ can be described as $Guard \to Update$, where $[\![Guard]\!] \subseteq Q_S$ and $[\![Update]\!] \subseteq Q_S \times Q_{S \cup \{y\}}$, defining the set of output actions $\{s \xrightarrow{y!v} t \mid s \in [\![Guard]\!] \wedge (s, t[y \mapsto v]) \in [\![Update]\!]\}$.
- Internal Tasks: $\mathbf{A}$
  - An internal task $A \in \mathbf{A}$ can be described as $Guard \to Update$, where $[\![Guard]\!] \subseteq Q_S$ and $[\![Update]\!] \subseteq Q_S \times Q_S$, defining the set of internal actions $\{s \xrightarrow{\varepsilon} t \mid s \in [\![Guard]\!] \wedge (s, t) \in [\![Update]\!]\}$.

Formalizing Examples

- Buffer = $(\{in\}, \{out\}, \{x\}, Init, \{\{A_i\}\}, \{\{A_o\}\}, \emptyset)$
  - $Init:$ $\{0,1,\text{null}\}$ $x :=$ null, $A_i:$ $x :=$ in, $A_o:$ $out := x;$ $x :=$ null
- AsyncInc = $(\emptyset, \emptyset, \{x, y\}, Init, \emptyset, \emptyset, \{A_x, A_y\})$
  - $Init:$ nat $x := 0;$ $y := 0,$ $A_x:$ $x := x + 1,$ $A_y:$ $y := y + 1$
- Merge = $(\{in_1, in_2\}, \{out\}, \{x_1, x_2\}, Init, \{\{A_i^1\}, \{A_i^2\}\}, \{\{A_o^1, A_o^2\}\}, \emptyset)$
  - $Init:$ queue(msg) $x_1 :=$ null; $x_2 :=$ null,
  - $A_i^1:$ $\neg$Full$(x_1) \rightarrow$ Enqueue(in_1, x_1), $A_i^2:$ $\neg$Full$(x_2) \rightarrow$ Enqueue(in_2, x_2)
  - $A_o^1:$ $\neg$Empty$(x_1) \rightarrow$ $out :=$ Dequeue(x_1), $A_o^2:$ $\neg$Empty$(x_2) \rightarrow$ $out :=$ Dequeue(x_2)

Executions of an Asynchronous Process

Interleaving Semantics

- An execution starts from an initial state.
- At every step, only one of the enabled tasks in the current state is nondeterministically chosen and executed.
  - The order in which different tasks are executed is totally unconstrained.
- Such semantics is called the *interleaving semantics*.
- eA finite execution of an asynchronous process $P = (I, O, S, Init, \mathscr{A}_I, \mathscr{A}_O, \mathbf{A})$ consists of a sequence

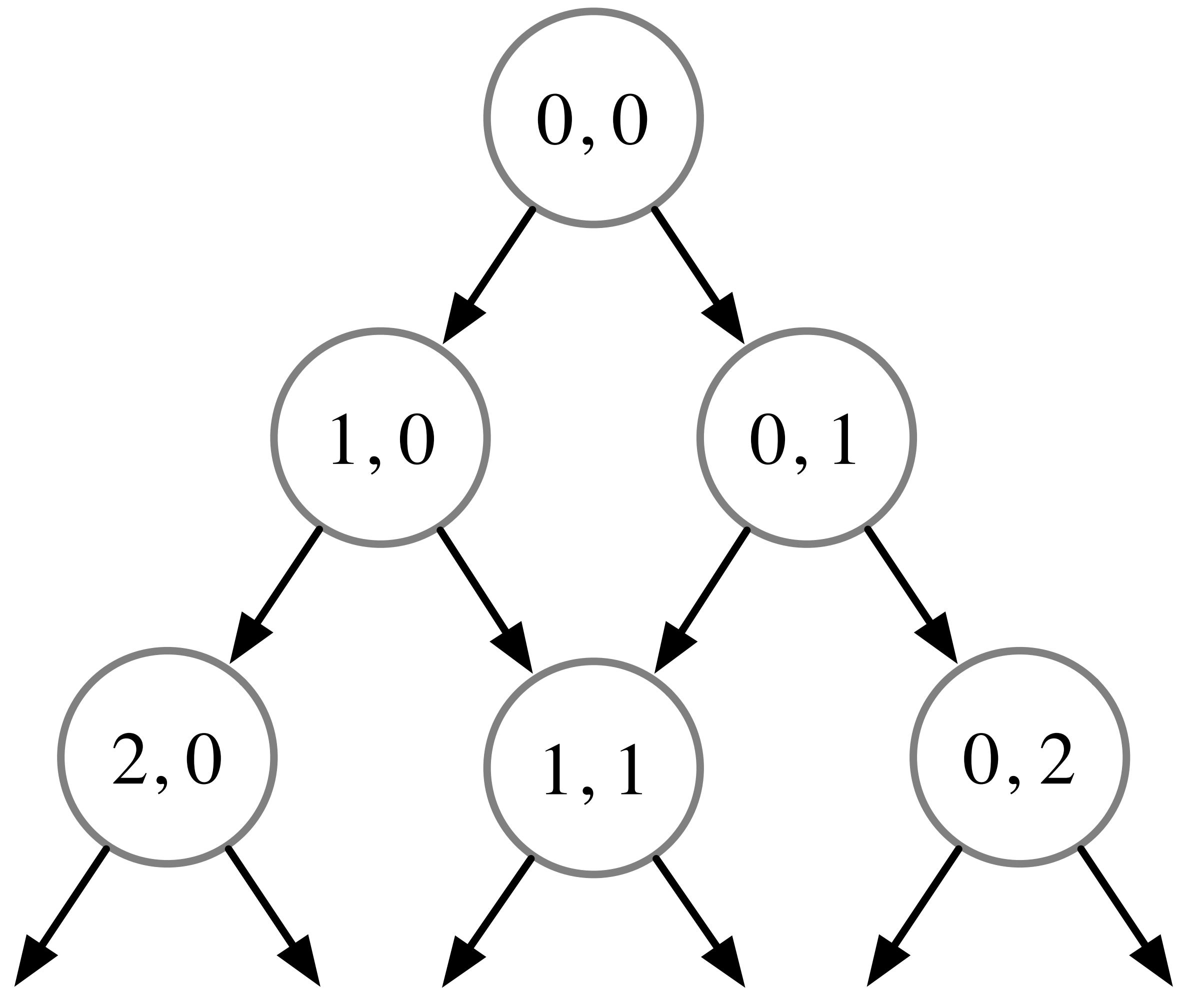
$$s_0 \xrightarrow{l_1} s_1 \xrightarrow{l_2} s_2 \xrightarrow{l_3} \cdots \xrightarrow{l_{k-1}} s_{k-1} \xrightarrow{l_k} s_k$$

  where $s_j \in Q_S$ for $0 \leq j \leq k$, $s_0 \in [\![Init]\!]$, and $s_{j-1} \xrightarrow{l_j} s_j$ is an input, output, or internal action for $1 \leq j \leq k$.

Executions of an Asynchronous Process

Examples

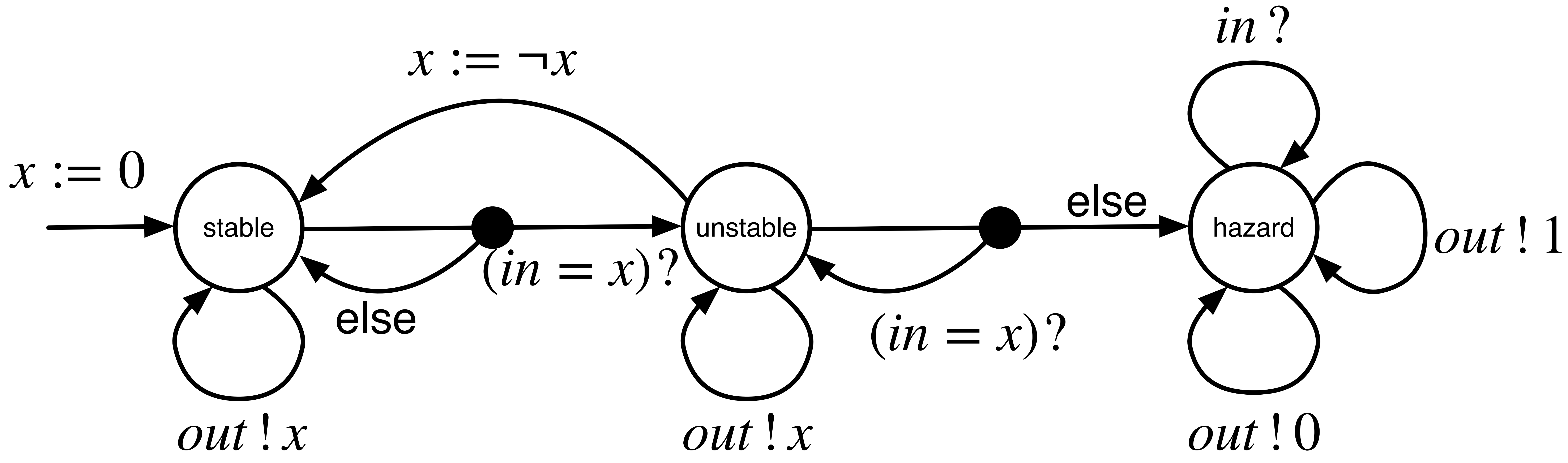
- Buffer
  - null $\xrightarrow{in?1}$ 1 $\xrightarrow{out?1}$ null $\xrightarrow{in?0}$ 0 $\xrightarrow{in?1}$ 1 $\xrightarrow{in?1}$ 1 $\xrightarrow{out?1}$ null
- AsyncInc
  - The figure on the right shows possible executions.



- Merge
  - (null, null) $\xrightarrow{in_1?0}$ ([0], null) $\xrightarrow{in_1?2}$ ([02], null) $\xrightarrow{in_2?5}$ ([02], [5]) $\xrightarrow{out?5}$ ([02], null) $\xrightarrow{in_2,3}$ ([02], [3]) $\xrightarrow{out,0}$ ([2], [3]) $\xrightarrow{out,3}$ ([2], null) $\xrightarrow{in_1,0}$ ([20], null)

Ex. AsyncNot

Description using an Extended-State Machine

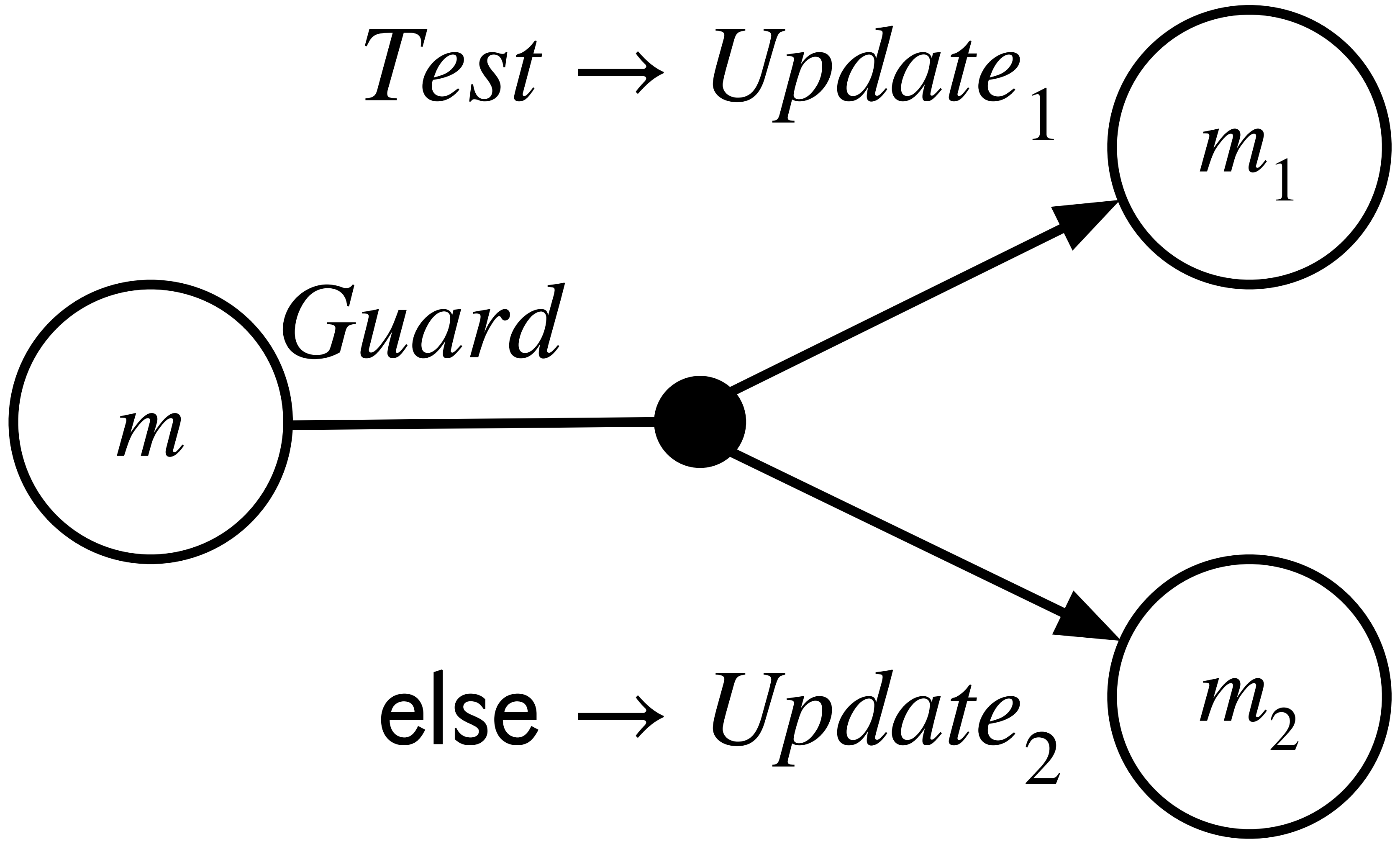


A possible execution

$$(\text{stable},0) \xrightarrow{out!0} (\text{stable},0) \xrightarrow{in!0} (\text{unstable},0) \xrightarrow{in!0} (\text{unstable},0) \xrightarrow{\varepsilon} (\text{stable},1) \xrightarrow{out!1} (\text{stable},1) \xrightarrow{out!1}$$

$$(\text{stable},1) \xrightarrow{in!1} (\text{unstable},1) \xrightarrow{in!0} (\text{hazard},1) \xrightarrow{out!0} (\text{hazard},1) \xrightarrow{out!1} (\text{hazard},1) \xrightarrow{in!0} (\text{hazard},1)$$

Conditional Mode-Switch



$\Longleftrightarrow$

$mode = m \land Guard \to$
if $Test$ then $\{Update_1;\ mode := m_1\}$
else $\{Update_2;\ mode := m_2\}$

- Note that $Guard$ cannot refer to any input channels but $Test$ can.
  - We cannot replace the conditional mode-switches in AsyncNot with ordinary mode-switches.

Ex. AsyncNot

Description using sets of tasks

$$\text{AsyncNot} = (\{in\}, \{out\}, \{mode, x\}, Init, \{\{A_1, A_3, A_6\}\}, \{\{A_2, A_4, A_7, A_8\}\}, \{A_5\})$$

$Init:\ \{\text{stable}, \text{unstable}, \text{hazard}\}\ mode := \text{stable};\ \ \text{bool}\ x := 0$

A_1: $mode = \text{stable} \rightarrow \text{if } in = x \text{ then } mode := \text{unstable}$

A_2: $mode = \text{stable} \rightarrow out\,!\,x$

A_3: $mode = \text{unstable} \rightarrow \text{if } \neg(in = x) \text{ then } mode := \text{hazard}$

A_4: $mode = \text{unstable} \rightarrow out\,!\,x$

A_5: $mode = \text{unstable} \rightarrow x := \neg x$

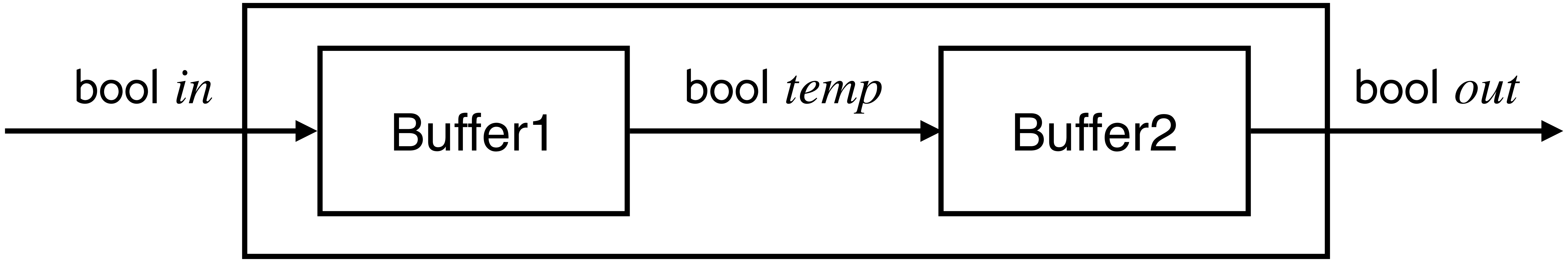
A_6: $mode = \text{hazard} \rightarrow \text{local bool } tmp;\ \ tmp := in$

A_7: $mode = \text{hazard} \rightarrow out\,!\,0$

A_8: $mode = \text{hazard} \rightarrow out\,!\,1$

Composing Asynchronous Processes

Ex. DoubleBuffer



- DoubleBuffer $= (\text{Buffer}[out \mapsto temp] \,|\, \text{Buffer}[in \mapsto temp]) \setminus temp$

- The composition of asynchronous processes can be defined similarly as in the case of synchronous reactive components.
  - Renaming of input/output channels
  - Parallel composition
  - Output hiding

Parallel Composition

- We write $P_1 | P_2$ to denote the parallel composition of two asynchronous processes P_1 and P_2.
- No name conflicts concerning state variables happen.
- The two sets of output channels are disjoint.
- An input channel of one process can be an input/output channel of the other.
  - We don't need to consider the problem of mutually cyclic await-dependencies.
    - ∵ Production of an output is a separate step from processing an input.
- The state of $P_1 | P_2$ can be expressed as (s_1, s_2) where s_1 and s_2 are the states of P_1 and P_2, respectively.

Combined Tasks of $P_1 \,|\, P_2$

Case 1: x is a common input channel of both P_1 and P_2

- If $s_1 \xrightarrow{x?v} t_1$ and $s_2 \xrightarrow{x?v} t_2$ are input actions of P_1 and P_2 respectively, then $(s_1, s_2) \xrightarrow{x?v} (t_1, t_2)$ is an input action of $P_1 \,|\, P_2$.
  - If $A_1 : Guard_1 \rightarrow Update_1$ and $A_2 : Guard_2 \rightarrow Update_2$ are respective input tasks of P_1 and P_2, both associated with x, then the combined input task A_{12} of $P_1 \,|\, P_2$ can be defined as $Guard_1 \wedge Guard_2 \rightarrow Update_1;\ Update_2$. The order of updates does not matter.
    - If $s_1 \in [\![Guard_1]\!]$, $s_2 \in [\![Guard_2]\!]$, $(s_1[x \mapsto v], t_1) \in [\![Update_1]\!]$, and $(s_2[x \mapsto v], t_2) \in [\![Update_2]\!]$ for an input value v, then $((s_1[x \mapsto v], s_2[x \mapsto v]), (t_1, t_2)) \in [\![Update_1;\ Update_2]\!]$.
  - If P_1 and/or P_2 have multiple input tasks associated with x, $P_1 \,|\, P_2$ has input tasks corresponding to all possible pairings of such tasks.

Combined Tasks of $P_1 \mid P_2$

Case 2: x is an output channel of P_1 and an input channel of P_2

- If $s_1 \xrightarrow{x!v} t_1$ is an output action of P_1 and $s_2 \xrightarrow{x?v} t_2$ is an input action of P_2, then $(s_1, s_2) \xrightarrow{x!v} (t_1, t_2)$ is an output action of $P_1 \mid P_2$.
  - If $A_1 : Guard_1 \to Update_1$ is an output task of P_1 and $A_2 : Guard_2 \to Update_2$ is an input task of P_2, both associated with x, then the combined input task A_{12} of $P_1 \mid P_2$ can be defined as $Guard_1 \wedge Guard_2 \to Update_1;\ Update_2$. The order of updates is significant.
    - If $s_1 \in [\![Guard_1]\!]$, $s_2 \in [\![Guard_2]\!]$, $(s_1, t_1[x \mapsto v]) \in [\![Update_1]\!]$, and $(s_2[x \mapsto v], t_2) \in [\![Update_2]\!]$ for a value v, then $((s_1, s_2[x \mapsto v]), (t_1[x \mapsto v], t_2)) \in [\![Update_1;\ Update_2]\!]$.
  - If P_1 and/or P_2 have multiple output/input tasks associated with x, $P_1 \mid P_2$ has output tasks corresponding to all possible pairings of such tasks.

Combined Tasks of $P_1 \mid P_2$

Case 3: x is a channel of P_1 but not of P_2

- If $s_1 \xrightarrow{x?v} t_1$ is an input action of P_1 and s is a state of P_2, then $(s_1, s) \xrightarrow{x?v} (t_1, s)$ is an input action of $P_1 \mid P_2$.
    - If A is an input task of P_1 associated with x which is not a channel of P_2, then A is also an input task of $P_1 \mid P_2$.
- If $s_1 \xrightarrow{x!v} t_1$ is an output action of P_1 and s is a state of P_2, then $(s_1, s) \xrightarrow{x!v} (t_1, s)$ is an output action of $P_1 \mid P_2$.
    - If A is an output task of P_1 associated with x which is not a channel of P_2, then A is also an output task of $P_1 \mid P_2$.
- Symmetric cases exist.

Combined Tasks of $P_1 | P_2$

Case 4: internal tasks

- If $s_1 \xrightarrow{\varepsilon} t_1$ is an internal action of P_1 and s is a state of P_2, then $(s_1, s) \xrightarrow{\varepsilon} (t_1, s)$ is an internal action of $P_1 | P_2$.
  - If $A_1 : Guard_1 \rightarrow Update_1$ is an internal task of P_1, then it is an internal task of $P_1 | P_2$.
- Symmetric case exists.

Parallel Composition

Ex. Buffer$[out \mapsto temp]$ | Buffer$[in \mapsto temp]$

bool in

$\{0{,}1{,}\text{null}\}\ x_1 := \text{null};\ x_2 := \text{null}$

$A_i:\ x_1 := in$

$A_t:\ (x_1 \neq \text{null}) \rightarrow \{temp := x_1;\ x_1 := \text{null};\ x_2 := temp\}$

$A_o:\ (x_2 \neq \text{null}) \rightarrow \{out := x_2;\ x_2 := \text{null}\}$

bool out

bool $temp$

- The output task of Buffer$[out \mapsto temp]$ and the input task of Buffer$[in \mapsto temp]$ are combined into an output task A_t.
  - The input channel $temp$ of Buffer$[in \mapsto temp]$ is no longer an input channel in the combined process.
  - The output channel $temp$ of Buffer$[out \mapsto temp]$ remains as an output channel in the combined process.

Asynchronous Process Composition

Definition (1/3)

- Let P_1 and P_2 be asynchronous processes such that $O_1 \cap O_2 = \emptyset$.
  - $P_1 = (I_1, O_1, S_1, Init_1, \{\mathbf{A}_x^1 \mid x \in I_1\}, \{\mathbf{A}_y^1 \mid y \in O_1\}, \mathbf{A}_1)$
  - $P_2 = (I_2, O_2, S_2, Init_2, \{\mathbf{A}_x^2 \mid x \in I_2\}, \{\mathbf{A}_y^2 \mid y \in O_2\}, \mathbf{A}_2)$
- The parallel composition $P_1 \mid P_2$ is the asynchronous process $P = (I, O, S, Init, \{\mathbf{A}_x \mid x \in I\}, \{\mathbf{A}_y \mid y \in O\}, \mathbf{A}\}$ defined by:
- $S = S_1 \cup S_2$, $O = O_1 \cup O_2$, $I = (I_1 \cup I_2) \setminus O$
- $Init = Init_1;\ Init_2$
- $\mathbf{A} = \mathbf{A}_1 \cup \mathbf{A}_2$

Asynchronous Process Composition

Definition (2/3)

- For each $x \in I$,

  (1) if $x \notin I_2$, then $\mathbf{A}_x = \mathbf{A}_x^1$;

  (2) if $x \notin I_1$, then $\mathbf{A}_x = \mathbf{A}_x^2$; and

  (3) if $x \in I_1 \cap I_2$, then for each $A_1 \in \mathbf{A}_x^1$ and $A_2 \in \mathbf{A}_x^2$, $\mathbf{A}_x$ contains the task described by $Guard_1 \wedge Guard_2 \rightarrow Update_1;\ Update_2$, where $Guard_1 \rightarrow Update_1$ is the description of A_1 and $Guard_2 \rightarrow Update_2$ is the description of A_2.

Asynchronous Process Composition

Definition (3/3)

- For each $y \in O$,

  (1) if $y \in O_1 \setminus I_2$, then $\mathbf{A}_y = \mathbf{A}_y^1$;

  (2) if $y \in O_2 \setminus I_1$, then $\mathbf{A}_y = \mathbf{A}_y^2$; and

  (3) if $y \in O_1 \cap I_2$, then for each $A_1 \in \mathbf{A}_y^1$ and $A_2 \in \mathbf{A}_y^2$, $\mathbf{A}_y$ contains the task described by $Guard_1 \wedge Guard_2 \to Update_1;\ Update_2$, where $Guard_1 \to Update_1$ is the description of A_1 and $Guard_2 \to Update_2$ is the description of A_2.

  (4) Symmetric case of (3)

Output Hiding

Ex. DoubleBuffer

- $\mathsf{DoubleBuffer} = (\mathsf{Buffer}[out \mapsto temp] \,|\, \mathsf{Buffer}[in \mapsto temp]) \setminus temp$
  - The output channel $temp$ of $\mathsf{Buffer}[out \mapsto temp] \,|\, \mathsf{Buffer}[in \mapsto temp]$ is hidden by applying $\setminus temp$.
- The output task A_t of $\mathsf{Buffer}[out \mapsto temp] \,|\, \mathsf{Buffer}[in \mapsto temp]$ becomes an internal task by replacing it with:

$$A_t : \; x_1 \neq \mathsf{null} \rightarrow \{\mathsf{local\ bool}\ temp; \; temp := x_1; x_1 := \mathsf{null}; x_2 := temp\}$$

  - In this task, $temp$ is now a local variable.

Safety Requirements

- We can define a transition system T for an asynchronous process P.
  - The set of state variables S of P is also the set of state variables in T.
  - The initialization specification $Init$ of P is also the initialization specification in T.
  - If $s \xrightarrow{l} t$ is an action in P, then $s \rightarrow t$ is a transition in T.
- The definitions of properties, (inductive) invariants, reachability for transition systems defined from SRCs can naturally extended for the transition systems defined above.
- The notions of safety monitors, enumerative and symbolic reachability can also be applied to the transition systems defined above.

Summary

- Asynchronous Model (1)
  - Asynchronous Process
  - Input/Output Channels, Input/Output/Internal Tasks, Enabled Tasks
  - Conditional Mode-Switch
  - Parallel Composition