Cyber-Physical Systems (CSC.T431)

Synchronous Model (3)

Instructor: Takuo Watanabe (Department of Computer Science)

Agenda

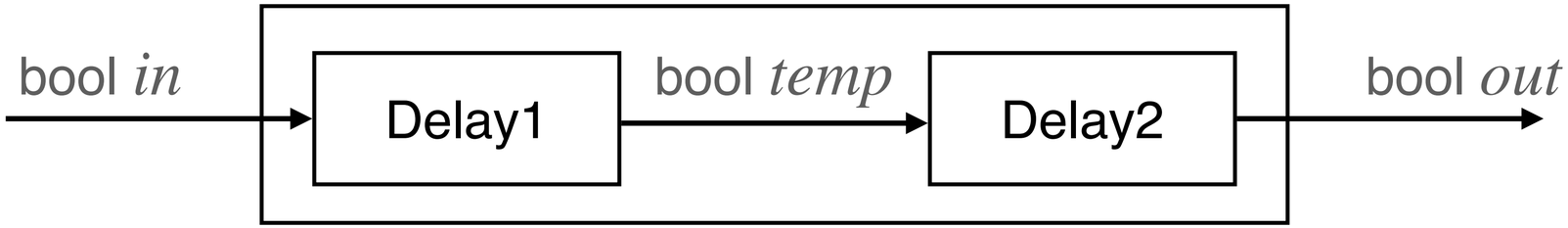
1. Synchronous Model (3)

Course Support & Material

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

Composition of Components

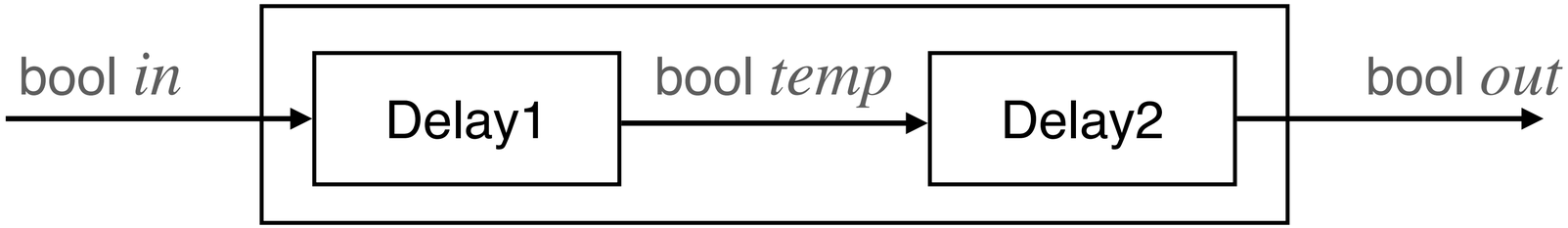
Ex. DoubleDelay from two Delays



- The DoubleDelay is a component with a Boolean input in and a Boolean output out, such that in the first two rounds the output is 0 and in every subsequent round n, the output equals the input in round $n - 2$.
- We can construct DoubleDelay by composing two Delay components as the block diagram above shows.

Composition of Components

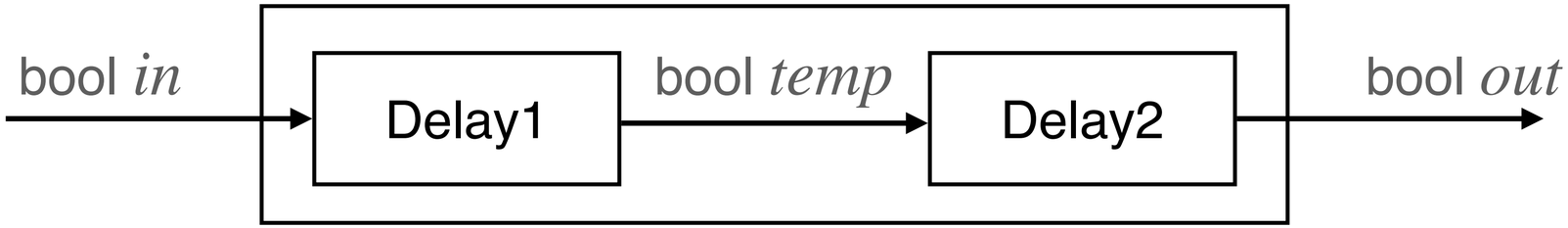
Three Operations on Composition



- Instantiation & Renaming
  - Delay1 (Delay2) is a copy of Delay except its output (input) variable is renamed to *temp*.
- Parallel Composition
  - Delay1 and Delay2 run in parallel and communicate synchronously via *temp*.
- Output Hiding
  - The variable *temp* is not exported to the outside of DoubleDelay.

Composition of Components

Three Operations on Composition



- DoubleDelay is textually defined as

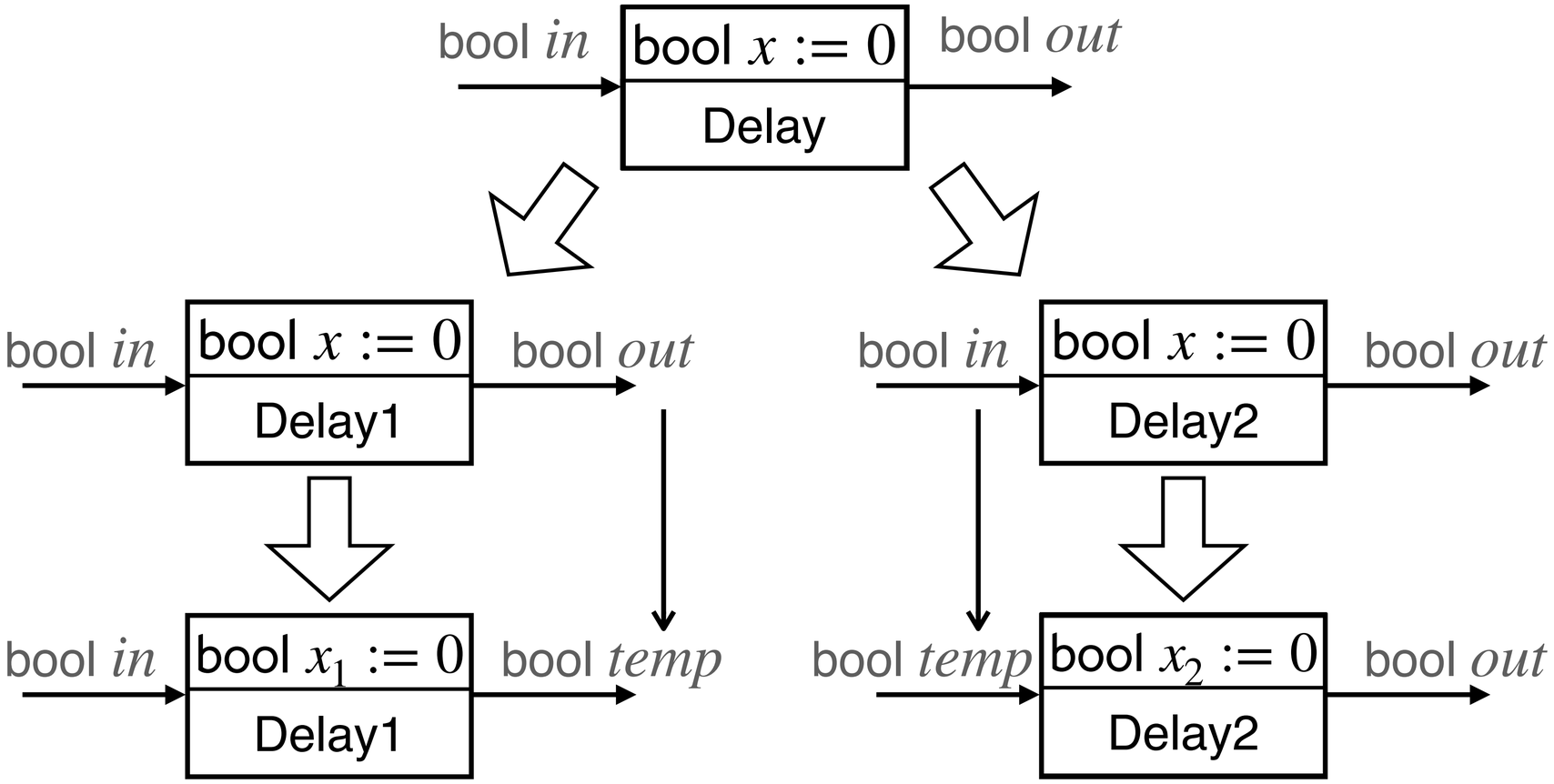
$$(\text{Delay}[out \mapsto temp] \;||\; \text{Delay}[in \mapsto temp]) \setminus temp$$

renaming — parallel composition — output hiding

Instantiation & Renaming

Instantiation

Renaming



Delay1 $=$ Delay$[out \mapsto temp]$

Delay2 $=$ Delay$[in \mapsto temp]$

Compatibility in Variable Names

- Two SRCs (Synchronous Reactive Components)
  $C_i = (I_i, O_i, S_i, Init_i, React_i)$ $(i = 1,2)$ are *compatible* if
  - $S_1 \cap (I_2 \cup O_2 \cup S_2) = \emptyset$ and $S_2 \cap (I_1 \cup O_1 \cup S_1) = \emptyset$,
  - $O_1 \cap O_2 = \emptyset$, and
  - $S_1 \cap S_2 = \emptyset$.

- $I_1 \cap I_2, I_1 \cap O_2, I_2 \cap O_1$ are not necessarily empty.
- If $x \in I_i$ and $y \in O_j$ $(i \neq j)$ are the same variable, then $\tau_x = \tau_y$.

Parallel Composition (1)

- $C_i = (I_i, O_i, S_i, Init_i, React_i)$ $(i = 1,2)$ are SRCs with compatible variable names. $C_1 \parallel C_2$ denotes the *parallel composition* (or the *synchronous product*) of C_1 and C_2.
- $C_1 \parallel C_2$ (say C hereafter) is also an SRC $(I, O, S, Init, React)$ where
  - $I = (I_1 \cup I_2) \setminus O$,
  - $O = O_1 \cup O_2$,
  - $S = S_1 \cup S_2$, and
  - $Init = Init_1; Init_2$ (or $Init_2; Init_1$).

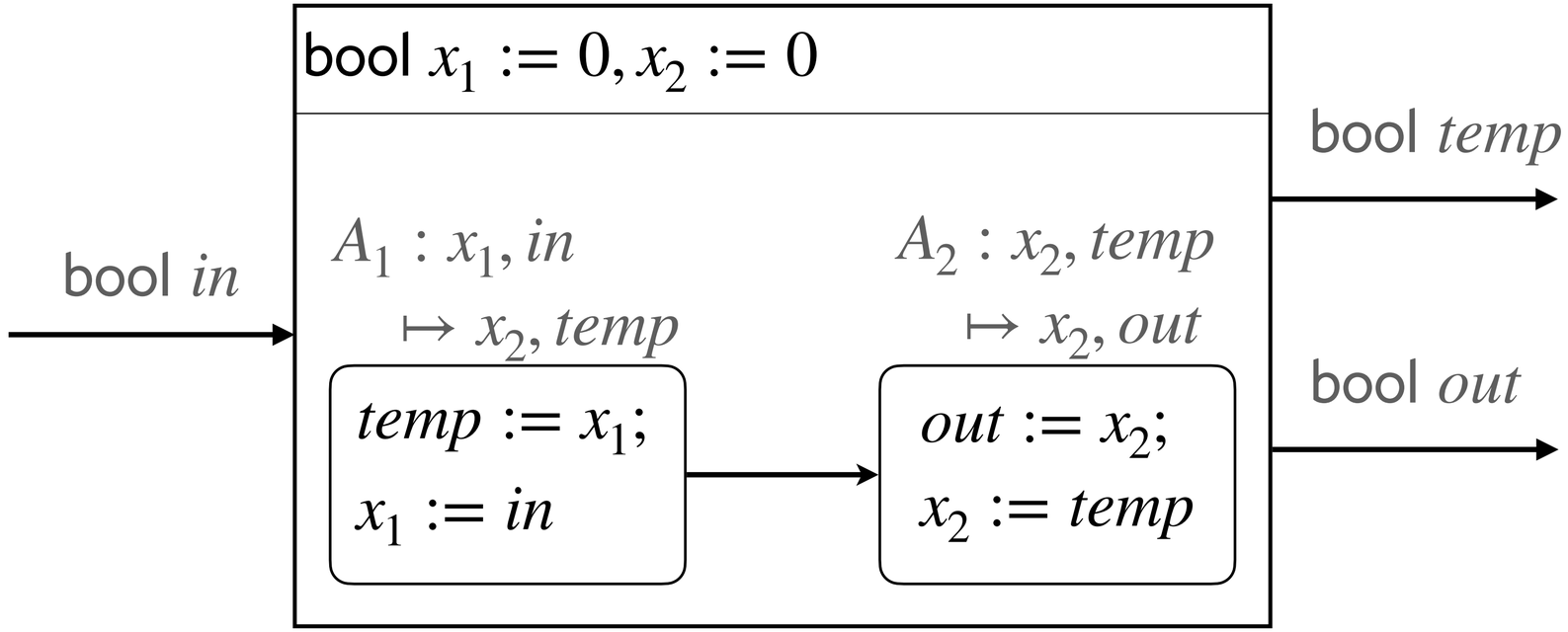
Parallel Composition (2)

How $React$ is defined

- No Communication: $I_1 \cap O_2 = \varnothing \wedge I_2 \cap O_1 = \varnothing$
  - Updates in $React_1$ and $React_2$ can be executed independently.
- 1-Way: $(I_1 \cap O_2 = \varnothing \wedge I_2 \cap O_1 \neq \varnothing) \vee (I_1 \cap O_2 \neq \varnothing \wedge I_2 \cap O_1 = \varnothing)$
  - Udates in $React$ can be realized by executing updates in $React_1$ first, then executing updates in $React_2$.
  - Ex. DoubleDelay
- 2-Way: $I_1 \cap O_2 \neq \varnothing \wedge I_2 \cap O_1 \neq \varnothing$
  - TBD

1-Way Communication

Ex. DoubleDelay

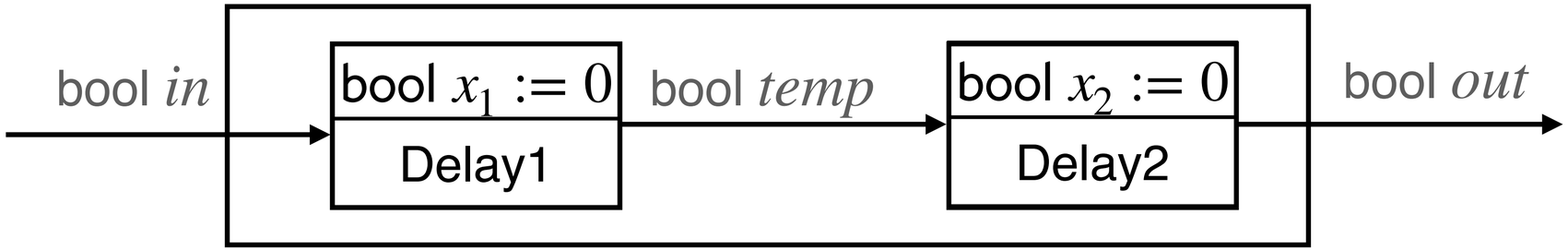


Reactions:

$$(0,0) \xrightarrow{0/(0,0)} (0,0);\ (0,0) \xrightarrow{1/(0,0)} (1,0);\ (0,1) \xrightarrow{0/(0,1)} (0,0);\ (0,1) \xrightarrow{1/(0,1)} (1,0);$$

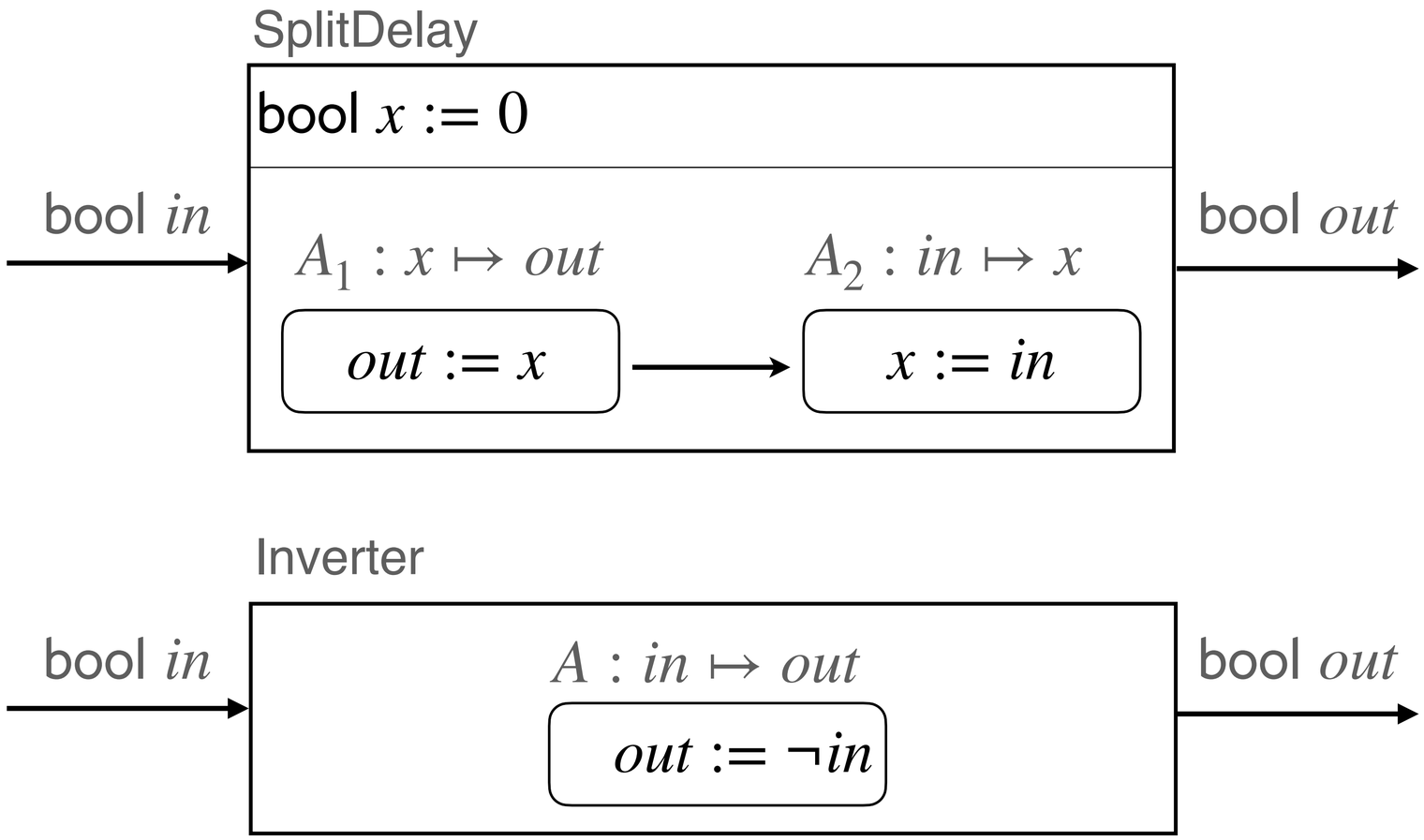
$$(1,0) \xrightarrow{0/(1,0)} (0,1);\ (1,0) \xrightarrow{1/(1,0)} (1,1);\ (1,1) \xrightarrow{0/(1,1)} (0,1);\ (1,1) \xrightarrow{1/(1,1)} (1,1)\,.$$

An Execution Scenario of DoubleDelay



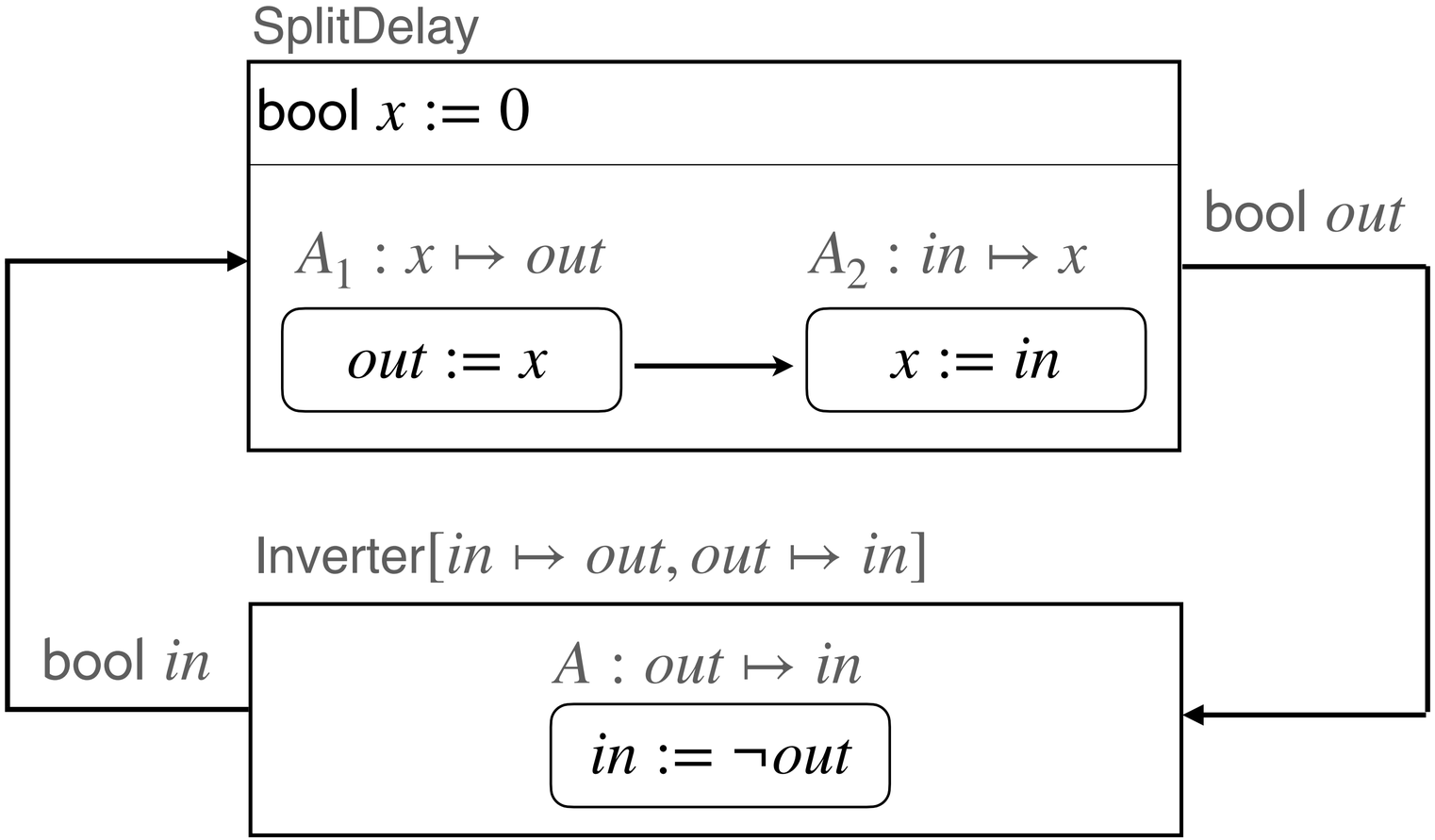
round	**init**	**1**	**2**	**3**	**4**	**5**	**6**
in		0	1	1	0	1	1
x_1'		0	0	1	1	0	1
x_1'	0	0	1	1	0	1	1
temp		0	0	1	1	0	1
x_2		0	0	0	1	1	0
x_2'	0	0	0	1	1	0	1
out		0	0	0	1	1	0

Ex. SplitDelay & Inverter



FeedBack Composition

Ex. of 2-Way Communication

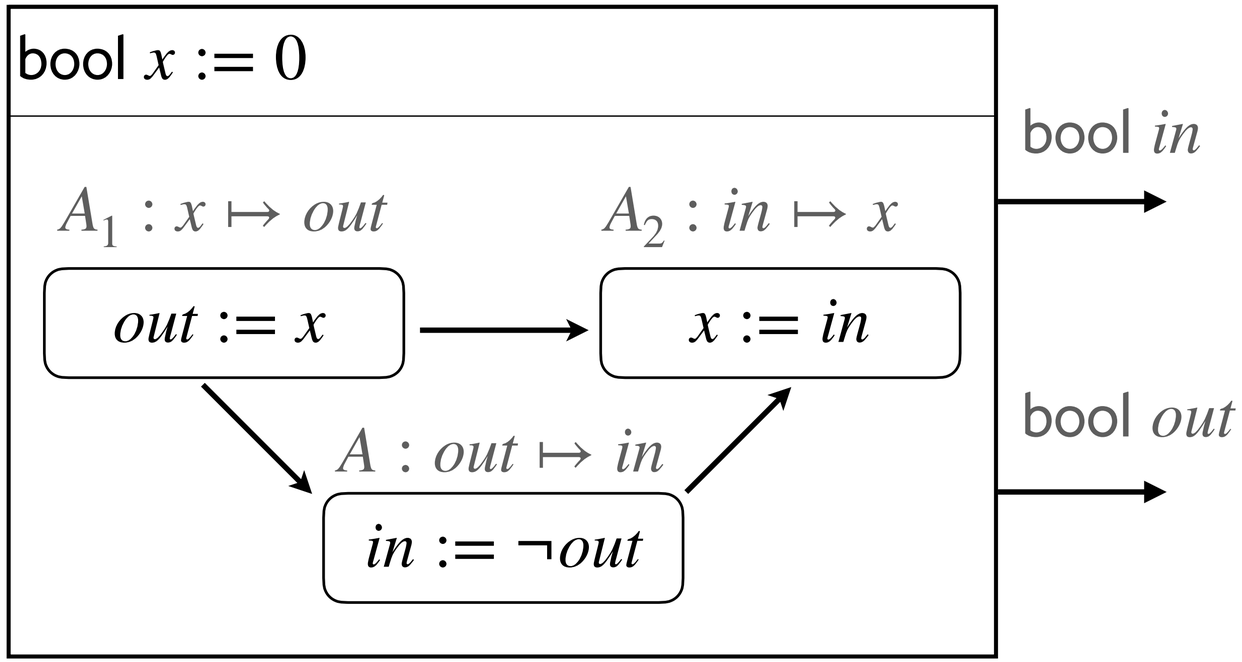


Composing Task Graphs

To construct the task graph of $C_1 \parallel C_2$

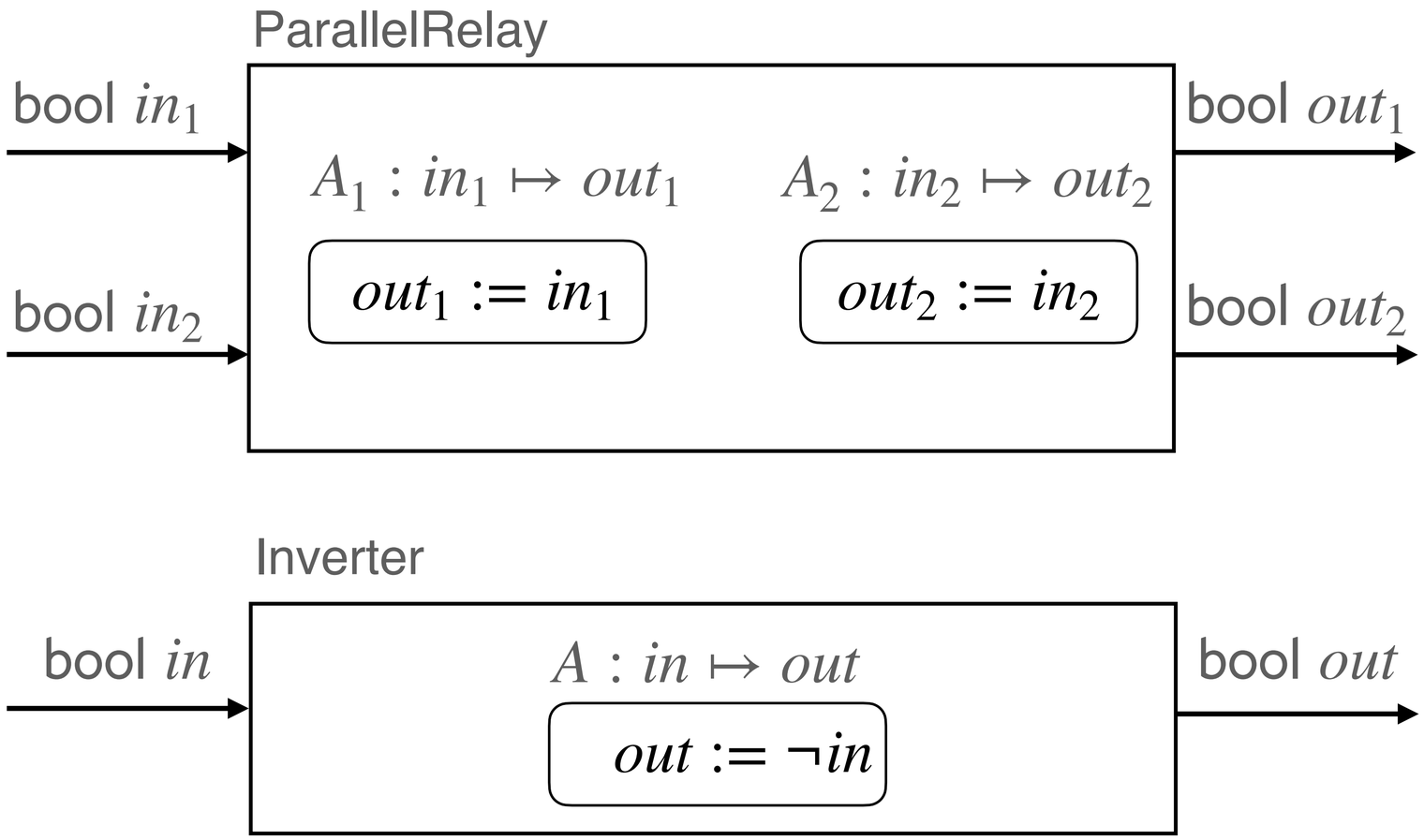
- *General rule for cross-component precedence edges*: Suppose that $i \neq j$. If $y \in O_j$ is a variable that is read by a task A in C_i, then add a precedence edge from A' to A where A' is the unique task in C_j that writes y.
- The precedence relation $\prec$ of $C_1 \parallel C_2$ is the union of $\prec_1$ and $\prec_2$, together with the cross-component edges according to the rule above.
- Ex.
  - The output variable in the renamed Inverter is read by the task A_2 in SplitDelay. The unique task in the renamed Inverter that write in is A. So we can add an edge from A to A_2.
  - Similarly, we can add an edge from A_1 to A.

SplitDelay ‖ Inverter$[in \mapsto out, out \mapsto in]$

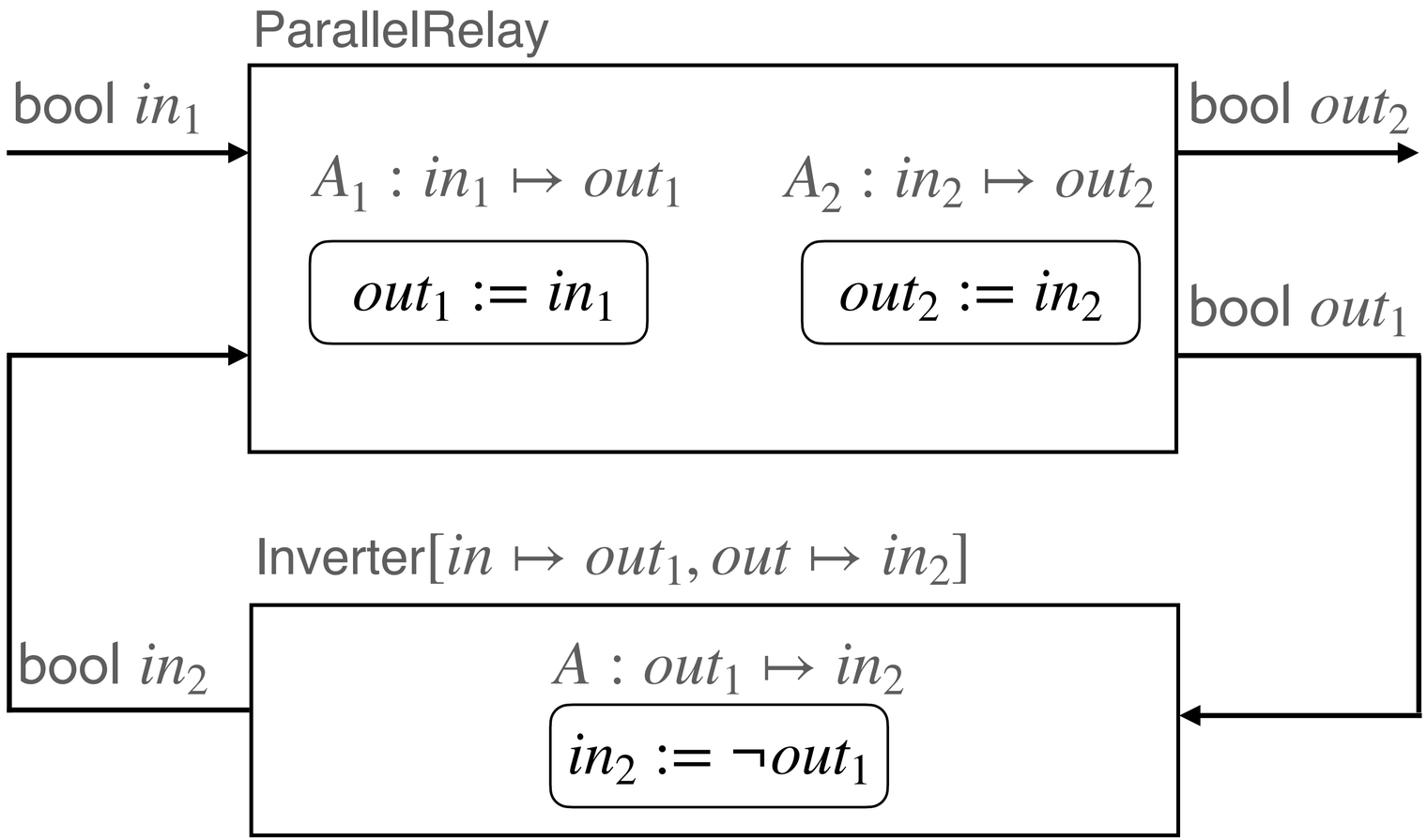


$$0 \xrightarrow{/(1,0)} 1 \xrightarrow{/(0,1)} 0 \xrightarrow{/(1,0)} 1 \xrightarrow{/(0,1)} 0 \xrightarrow{/(1,0)} \cdots$$

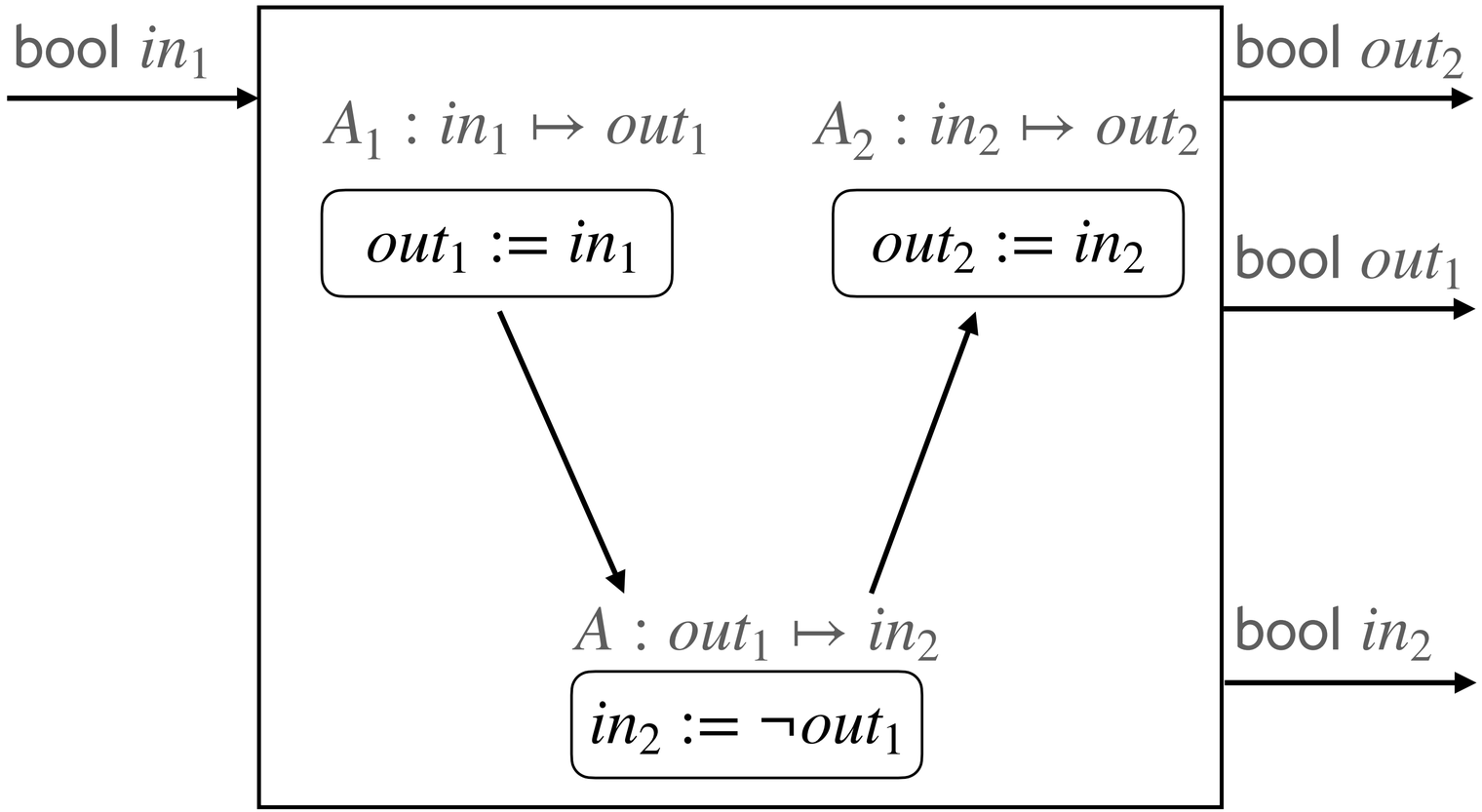
Ex. ParallelRelay & Inverter



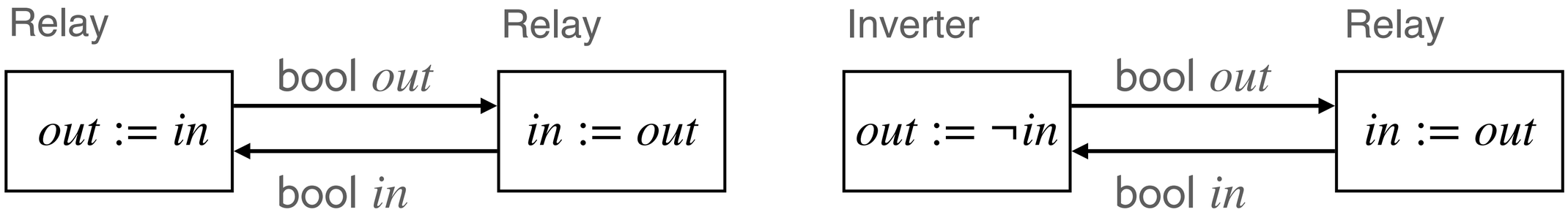
Ex. Parallel Composition



ParallelRelay || Inverter$[in \mapsto out_1, out \mapsto in_2]$



Cycles in Await Dependencies



- Relay ‖ Relay$[in \mapsto out, out \mapsto in]$
  - Both $S_\varnothing \xrightarrow{/(0,0)} S_\varnothing \xrightarrow{/(0,0)} S_\varnothing \xrightarrow{/(0,0)} \cdots$ and $S_\varnothing \xrightarrow{/(1,1)} S_\varnothing \xrightarrow{/(1,1)} S_\varnothing \xrightarrow{/(1,1)} \cdots$ are OK?
- Inverter ‖ Relay$[in \mapsto out, out \mapsto in]$
  - No valid computation

Component Compatibility

- C_1, C_2 : synchronous reactive components
  - $C_1 = (I_1, O_1, S_1, Init_1, React_1)$, $\succ_1 \subseteq O_1 \times I_1$: await-dependency
  - $C_2 = (I_2, O_2, S_2, Init_2, React_2)$, $\succ_2 \subseteq O_2 \times I_2$: await-dependency

- C_1 and C_2 are compatible if
  1. the set O_1 and O_2 are disjoint, and
  2. the relation $\succ_1 \cup \succ_2$ is acyclic.

Interface

- Let $C = (I, O, S, Init, React)$ be a component and $\succ \in O \times I$ is the await-dependency of C.
- We say that $(I, O, \succ)$ is the *interface* of C.

- To check the compatibility of two components, we only need their interfaces.

Proposition

Await compatibility implies acyclic product task graph

Let C_1 and C_2 be compatible reactive components. Then the task graph over the set of tasks in C_1 and C_2 obtained by retaining the precedence edges in the individual components and adding cross-component edges from a task A_1 of one component to a task A_2 of another component whenever A_1 writes a variable read by A_2, is acyclic.

In short, If $\prec_1 \cup \prec_2$ is acyclic, then the task graph of $C_1 \parallel C_2$ is acyclic.

Proof Direction

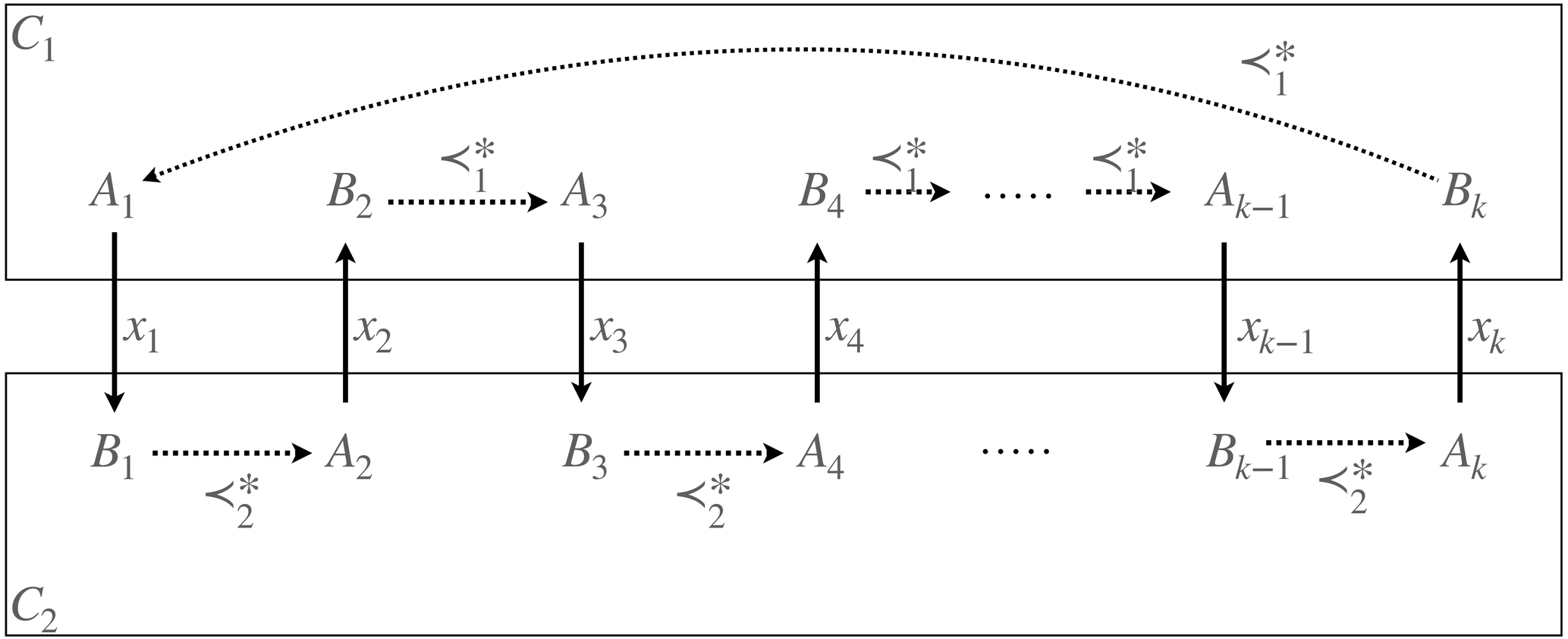
- C_1, C_2 : compatible components
- $\prec_1$, $\prec_2$: precedence relations over the task sets of C_1 and C_2
- $\succ_1$, $\succ_2$: input/output await dependencies of C_1 and C_2

- We show that if the combined task graph (task graph of $C_1 \parallel C_2$) contains a cycle, then $\succ$ ($= \succ_1 \cup \succ_2$) has a cycle.

- Notation: $A \prec^* B \stackrel{\text{def}}{=} A = B \vee A \prec^+ B$

Proof

Suppose that the combined task graph has a cycle. It should look like this.



cross-component edges

For $j = 1,\ldots,k$, let (A_j, B_j) be a pair of tasks that corresponds to a cross-component edge in the cycle and x_j be a variable written by A_j and read by B_j. Suppose that $p = (j+1) \mod 2 + 1$ and $q = (j+1) \mod 2 + 2$ and A_j and B_j respectively belong to components C_p and C_q. This implies $x_j \in O_p$ and $x_j \in I_q$. Thus, $x_j \succ_p x_{j-1}$ $(j = 2,\ldots,k)$ and $x_1 \succ_1 x_k$. Clearly this forms a cycle. ∎

Component Composition

- $C_i = (I_i, O_i, S_i, Init_i, React_i)\ (i = 1,2)$: compatible SRCs
  - $React_i$ is given using local variables L_i by a task graph with the set $\mathbf{A}_i$ of tasks and the precedence relation $\prec_i$.
- $C_1 \parallel C_2$ is a SRC $C = (I, O, S, Init, React)$ such that
  - $I = (I_1 \cup I_2) \setminus O$, $O = O_1 \cup O_2$, $S = S_1 \cup S_2$
  - If $x \in S_i$, its initialization in $Init$ is given by $Init_i$ $(i = 1,2)$.
  - $React$ uses local variables $L_1 \cup L_2$ and is given by a task graph such that (1) the set of tasks $\mathbf{A}_1 \cup \mathbf{A}_2$, and (2) the precedence relation is $\prec_1 \cup \prec_2$ and task pairs (A_1, A_2), such that A_1 and A_2 are tasks of different components with some variable occurring in both the write-set of A_1 and the read-set of A_2.

Properties of Parallel Components

- Let C_1, C_2 and C_3 be SRCs in which any pair of them are compatible. The following properties hold.

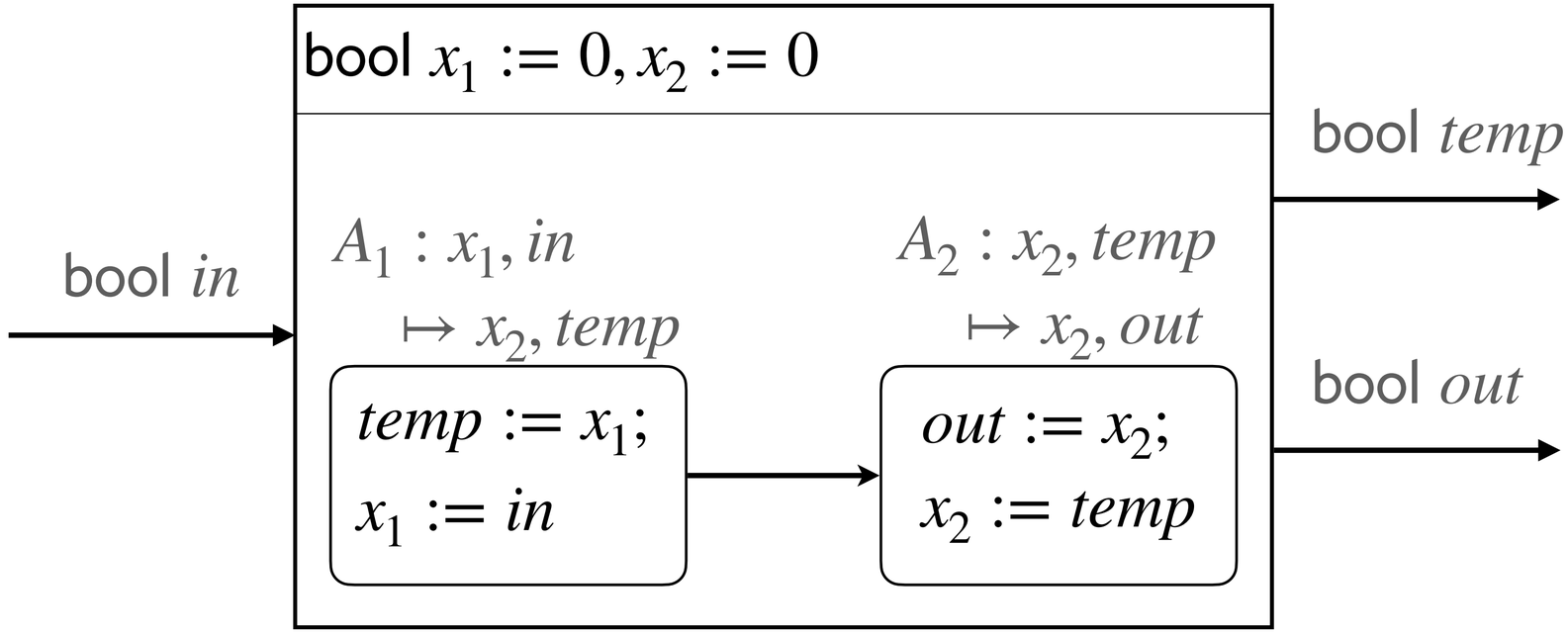
- Commutativity: $C_1 \parallel C_2 = C_2 \parallel C_1$
- Associativity: $C_1 \parallel (C_2 \parallel C_3) = (C_1 \parallel C_2) \parallel C_3$
- If both C_1 and C_2 are finite-state, then so is $C_1 \parallel C_2$.
- If C_1 has n_1 states and C_2 has n_2 states, then $C_1 \parallel C_2$ has $n_1 \times n_2$ states.
- If all the tasks of C_1 and C_2 are deterministic, then $C_1 \parallel C_2$ is deterministic.
- If all the tasks of C_1 and C_2 are input-enabled, then $C_1 \parallel C_2$ is input-enabled.

Output Hiding

- Let $C = (I, O, S, Init, React)$ is an SRC and $y \in O$.
- $C \setminus y = (I, O \setminus \{y\}, S, Init, React_{(y)})$, where $React_{(y)}$ is the same as $React$ except that it uses y as a local variable.
- $(C \setminus x) \setminus y = (C \setminus y) \setminus x$ holds. So we can write $C \setminus \{x, y\}$.

DoubleDelay (revisited)

Delay[$out \mapsto temp$] ‖ Delay[$in \mapsto temp$]



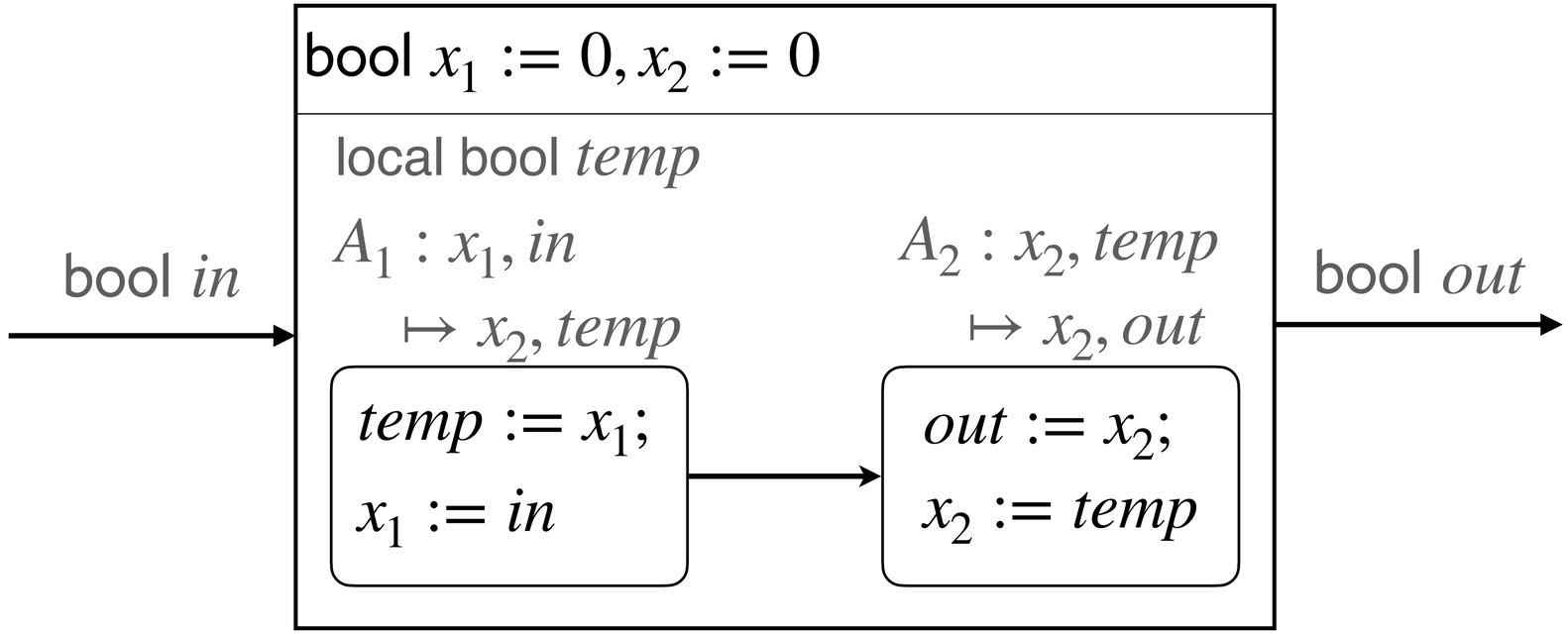
Reactions:

$$(0,0) \xrightarrow{0/(0,0)} (0,0);\ (0,0) \xrightarrow{1/(0,0)} (1,0);\ (0,1) \xrightarrow{0/(0,1)} (0,0);\ (0,1) \xrightarrow{1/(0,1)} (1,0);$$

$$(1,0) \xrightarrow{0/(1,0)} (0,1);\ (1,0) \xrightarrow{1/(1,0)} (1,1);\ (1,1) \xrightarrow{0/(1,1)} (0,1);\ (1,1) \xrightarrow{1/(1,1)} (1,1)\,.$$

DoubleDelay (revisited)

(Delay[$out \mapsto temp$] ‖ Delay[$in \mapsto temp$]) \ $temp$



Reactions:

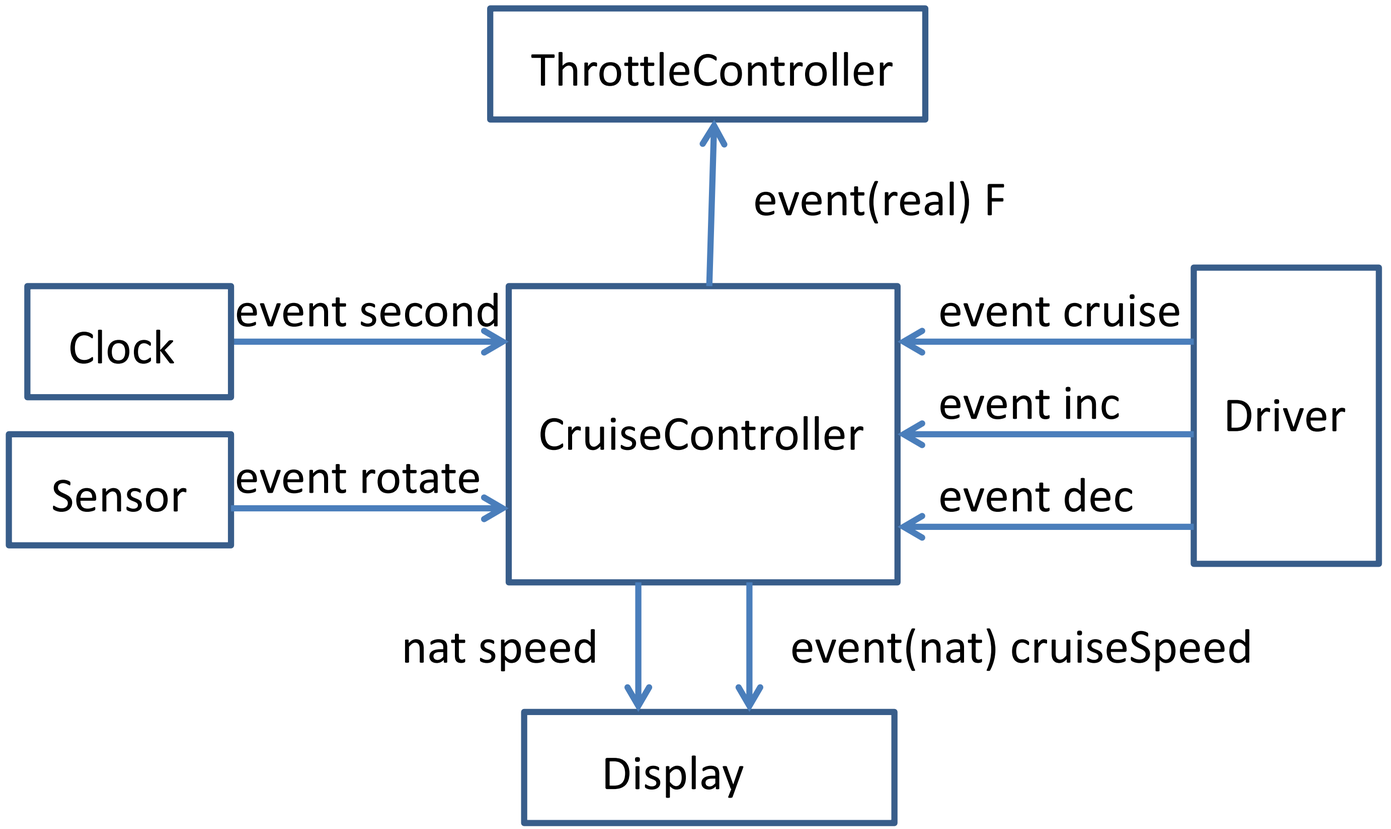
$$(0,0) \xrightarrow{0/0} (0,0);\ (0,0) \xrightarrow{1/0} (1,0);\ (0,1) \xrightarrow{0/1} (0,0);\ (0,1) \xrightarrow{1/1} (1,0);$$

$$(1,0) \xrightarrow{0/0} (0,1);\ (1,0) \xrightarrow{1/0} (1,1);\ (1,1) \xrightarrow{0/1} (0,1);\ (1,1) \xrightarrow{1/1} (1,1)\,.$$

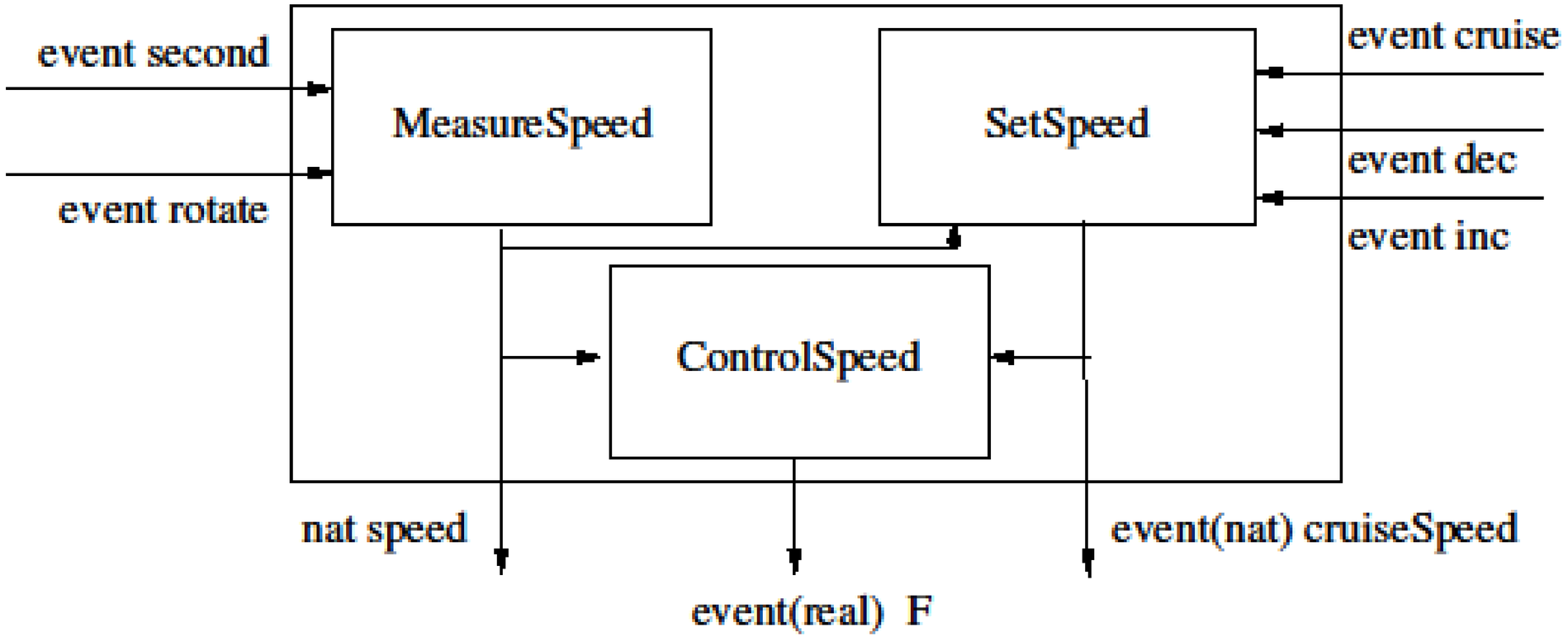
Top-Down Design of SRCs

- Starting point: Inputs and outputs of desired design C
- Models/assumptions about the environment in which C operates
- Informal/formal description of desired behavior of C
- Example: CruiseController

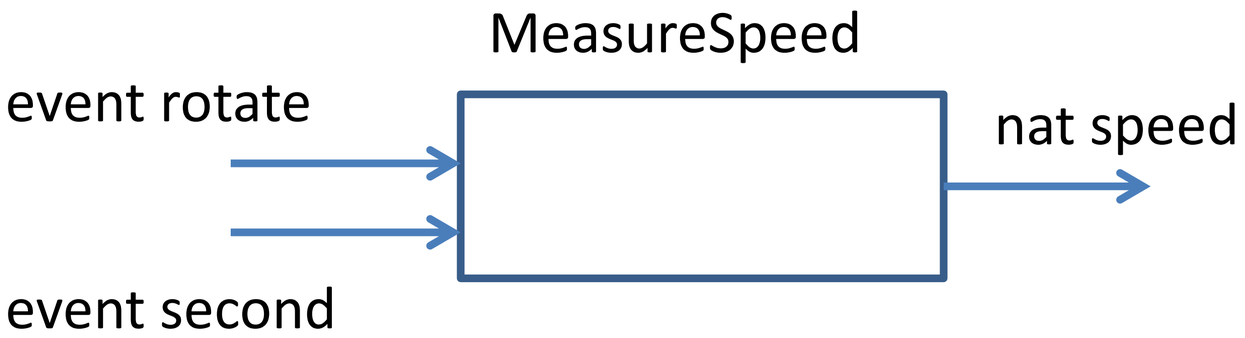
Ex. CruiseController



Decomposing CruiseController

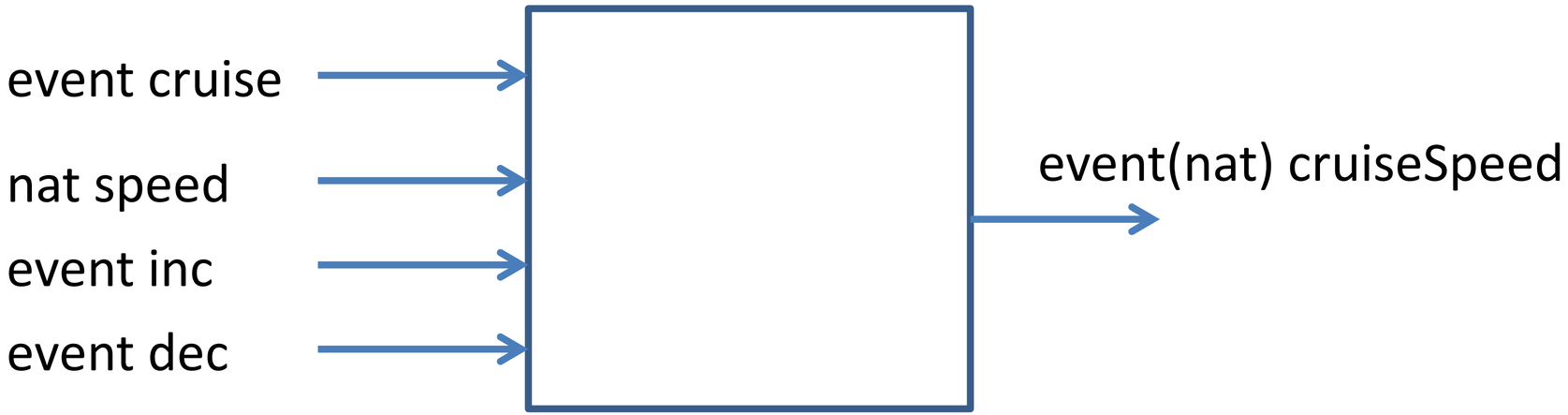


Tracking Speed



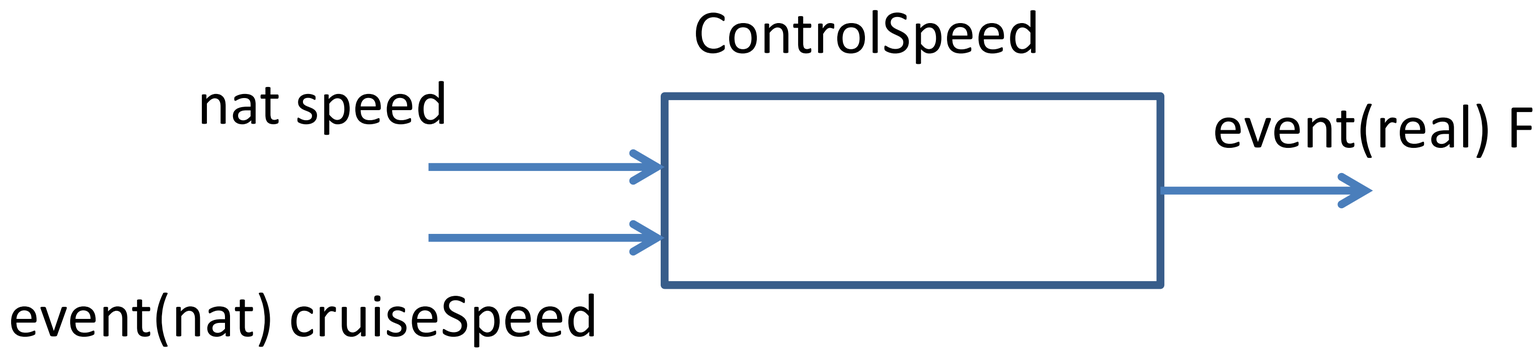
- Inputs: Events rotate and second
- Output: current speed
- Computes the number of rotate events per second

Tracking Cruise Settings



- Inputs from the driver: Commands to turn the cruise-control on/off and to increment/decrement desired cruising speed from driver
- Input: Current speed, Output: Desired cruising speed
- What assumptions can we make about simultaneity of events?
- Should we include safety checks to keep desired speed within bounds?

Controlling Speed



- Inputs: Actual speed and desired speed
- Output:Make Pressure on the throttle
- Goal: actual speed equal to the desired speed (while maintaining key physical properties such as stability)
- Design relies on theory of dynamical systems (Chapter 6)

Summary

- Synchronous Model (3)
  - Composition of Components
    - Instantiation and Renaming, Compatibility in Variable Names
    - Parallel Composition
      - Composition of Task Graphs, Cross-Components Edges
      - Compatibility, Interface
    - Output Hiding
  - Top-Down Design of SRCs
    - Ex. CruiseController