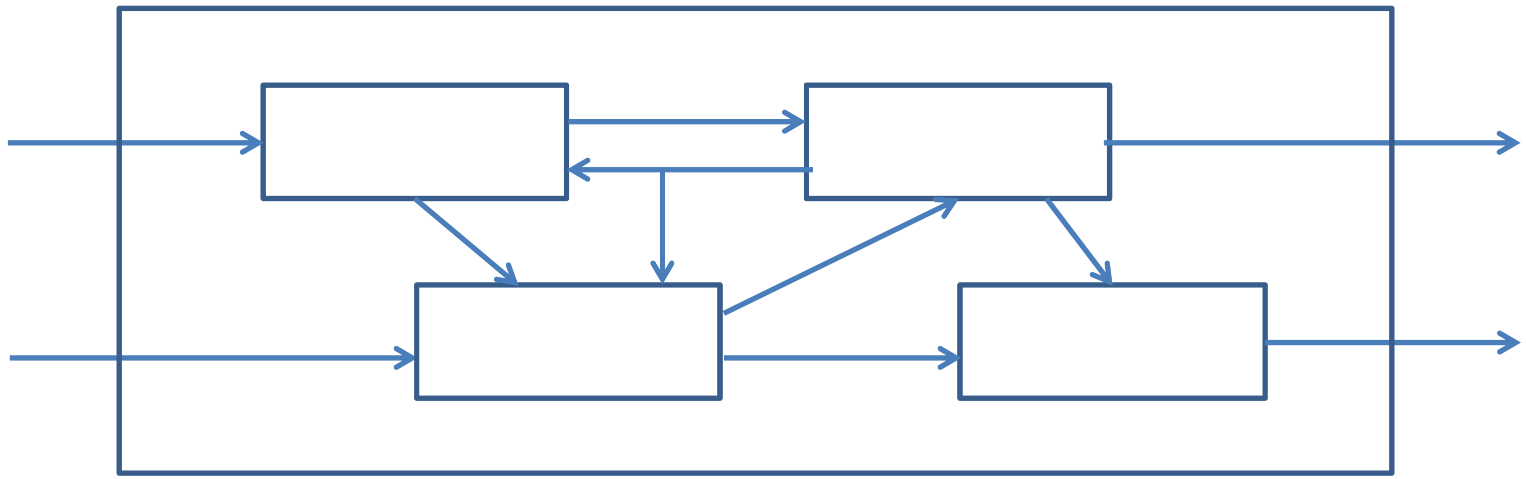
CS:5980
Foundations of Embedded Systems

The Synchronous Model
Part II

Copyright 2014-26, Rajeev Alur and Cesare Tinelli.
Created by Cesare Tinelli at the University of Iowa from notes originally developed by Rajeev Alur at the University of Pennsylvania. These notes are copyrighted materials and may not be used in other course settings outside of the University of Iowa in their current form or modified form without the express written permission of one of the copyright holders. During this course, students are prohibited from selling notes to or being paid for taking notes by any person or commercial firm without the express written permission of one of the copyright holders.

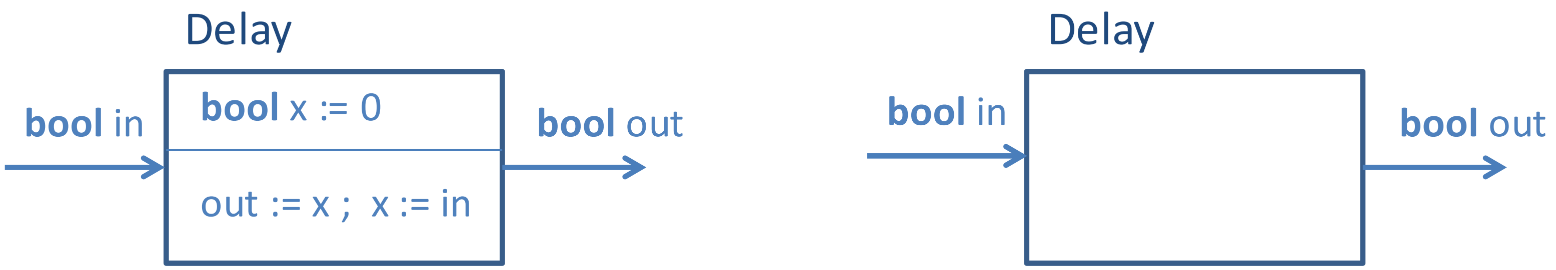
Block Diagrams



Structured modeling

- How do we build complex models from simpler ones?
- What are basic operations on components?

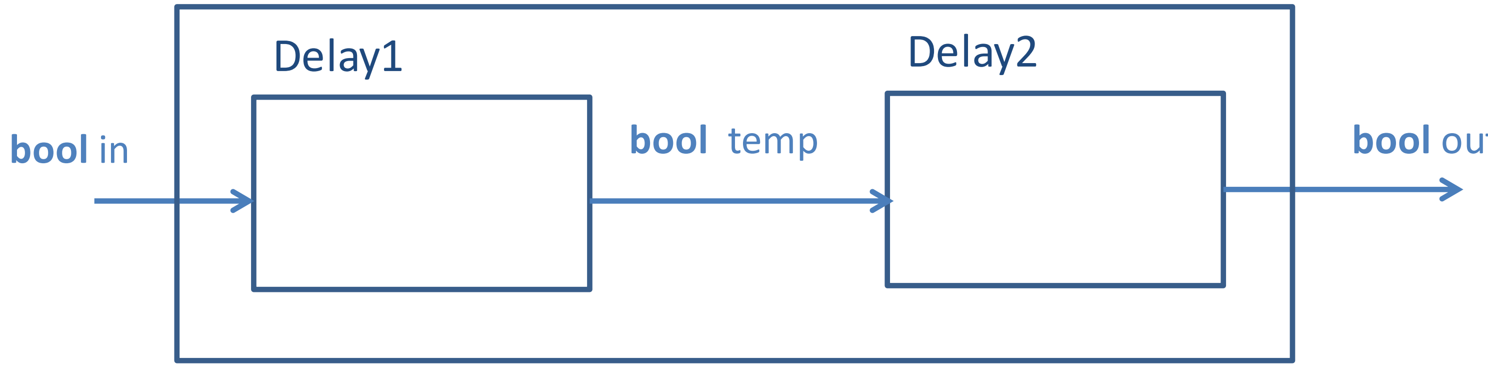
DoubleDelay



Design a component with

- **Input:** bool in
- **Output:** bool out
- **Behavior:** out in round n should equal in in round n-2

DoubleDelay



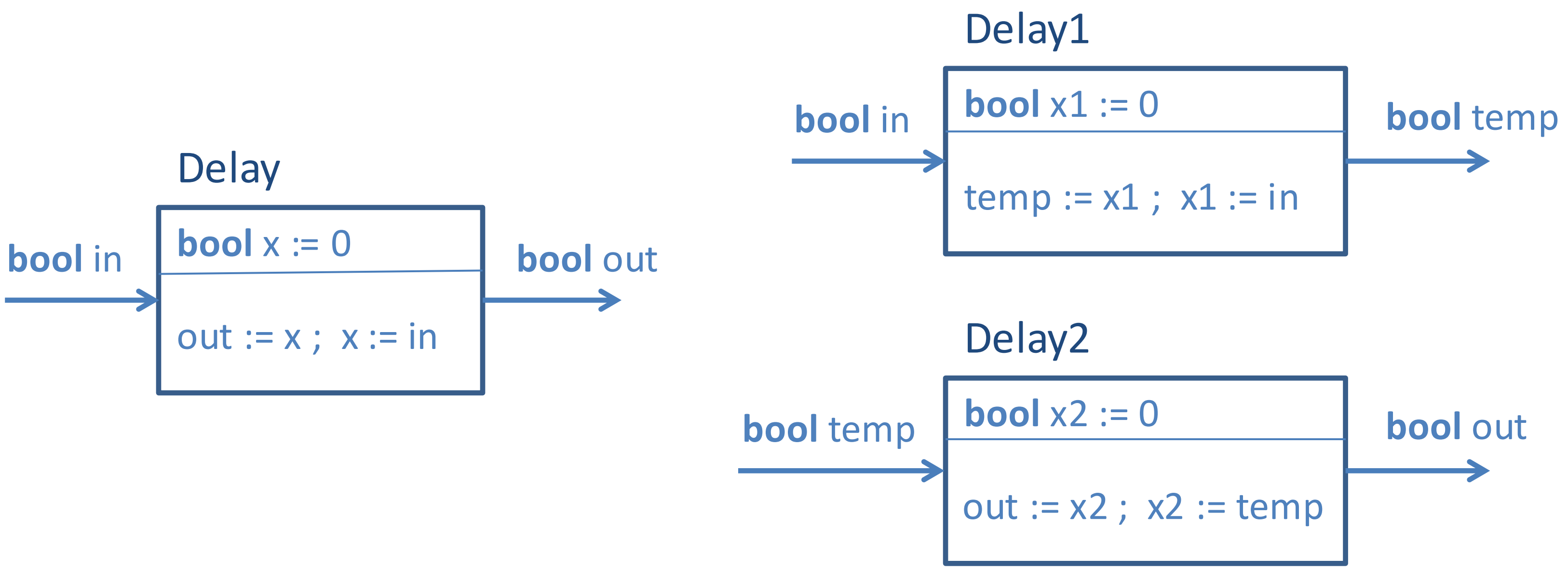
Instantiation: Create two instances of Delay

- output of Delay1 = input of Delay2 = variable temp

Parallel composition/product: Achieve concurrent execution of Delay1 and Delay2

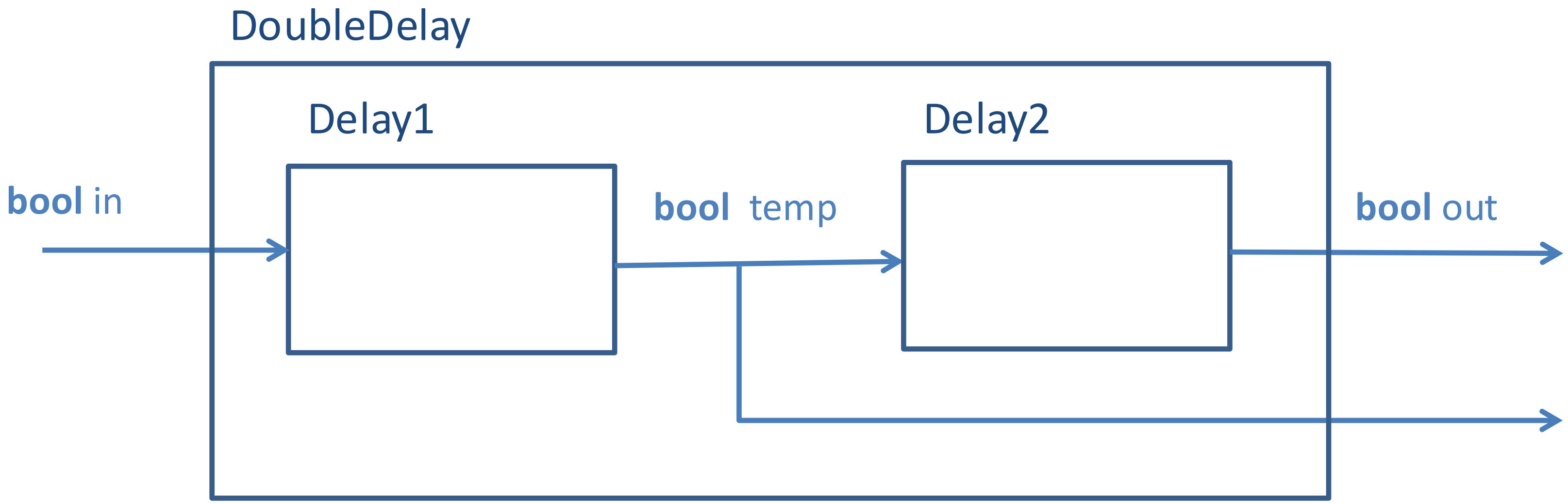
Encapsulation: Hide variable temp

Instantiation / Renaming



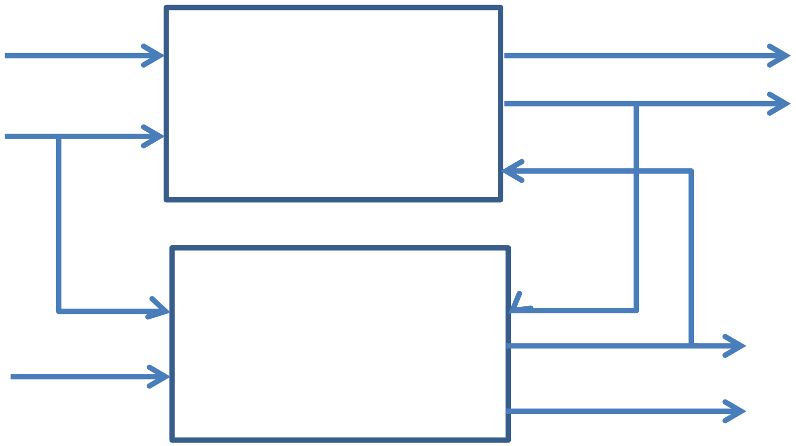
- Delay1 = Delay[out ↦ temp]
  - Explicit renaming of input/output variables
  - Implicit renaming apart of state variables
  - Definition $(I_1, O_1, S_1, Init_1, React_1)$ of Delay1 derived from Delay
- Delay2 = Delay[in ↦ temp]

Parallel Composition (or Product)



- DoubleDelay = Delay1 || Delay2
  - Two components execute concurrently
- When can two components be composed?
- How to define parallel composition precisely?
  - Input/output/state variables, initialization, and reaction description of composite defined in terms of components
  - Can be viewed as an algorithm for compilation

Compatibility of Components C1 and C2



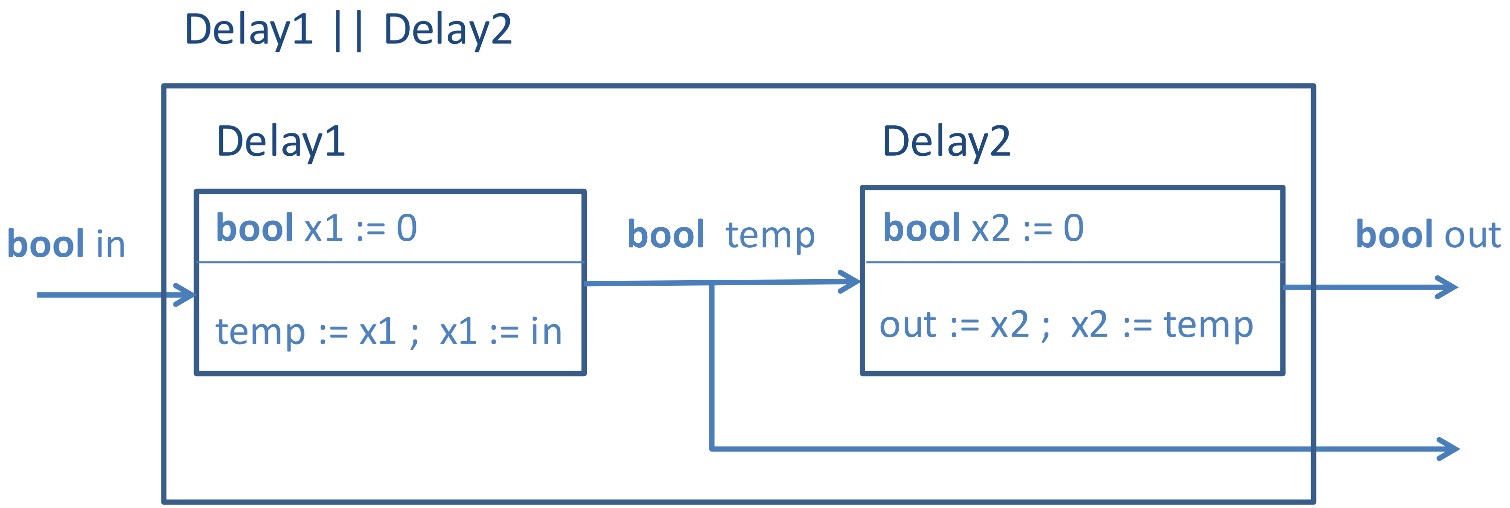
Allowed:

- Shared input variables
- Output variable of one is input variable of the other

Disallowed:

- Shared output variables
  - A unique component must be responsible for values of any given variable
- Shared state variables
  - However, state variables can always be renamed to avoid conflicts

Outputs of Product

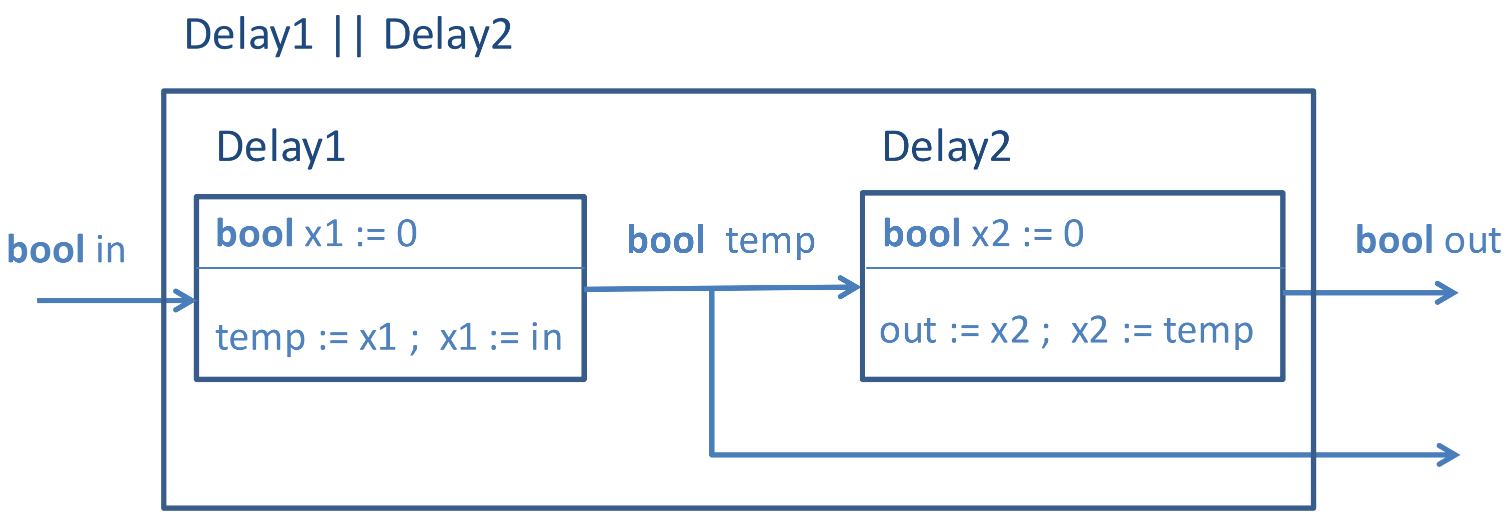


The output variables of Delay1 || Delay2 are { temp, out }

- By default, every output of product is available to outside world

If C_1 has output vars O_1 and C_2 has output vars O_2
then $C_1 \,||\, C_2$ has output vars $O_1 \cup O_2$

Inputs of Product



The input variables of Delay1 || Delay2 are { in }

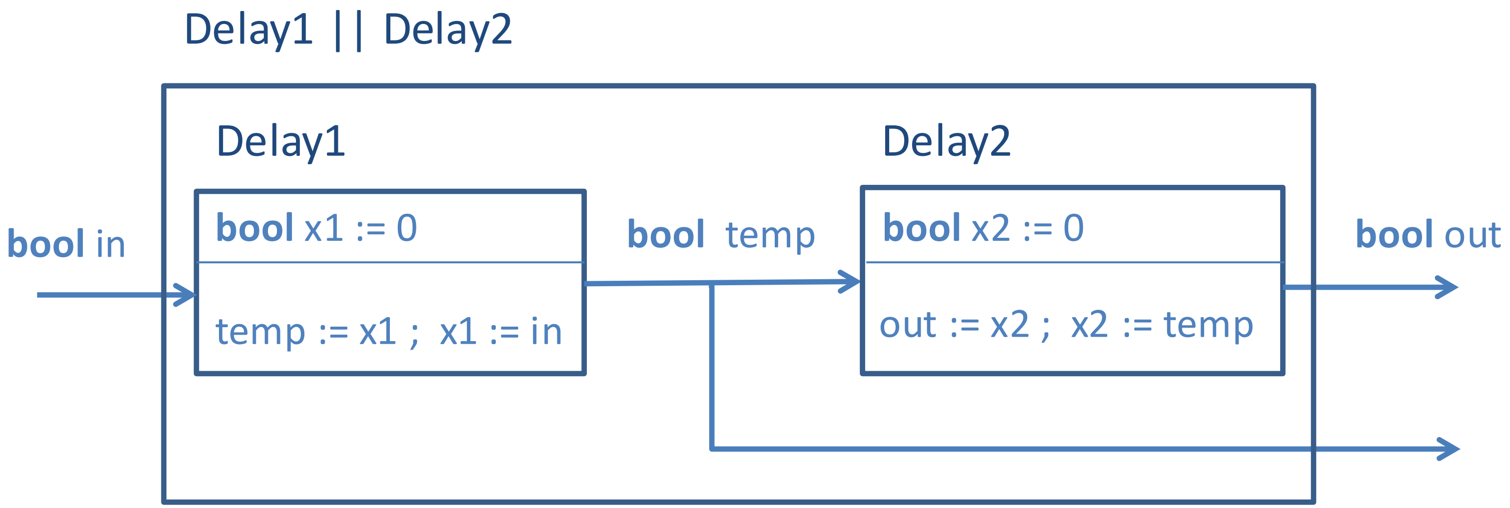
- Even though temp is input of Delay2, it is not an input of product

If C_1 has input vars I_1 and C_2 has input vars I_2

then $C_1 \,||\, C_2$ has input vars $(I_1 \cup I_2) \setminus (O_1 \cup O_2)$

- A variable is an input of the product iff
  it is an input of one of the components, and not an output of the other

States of Product



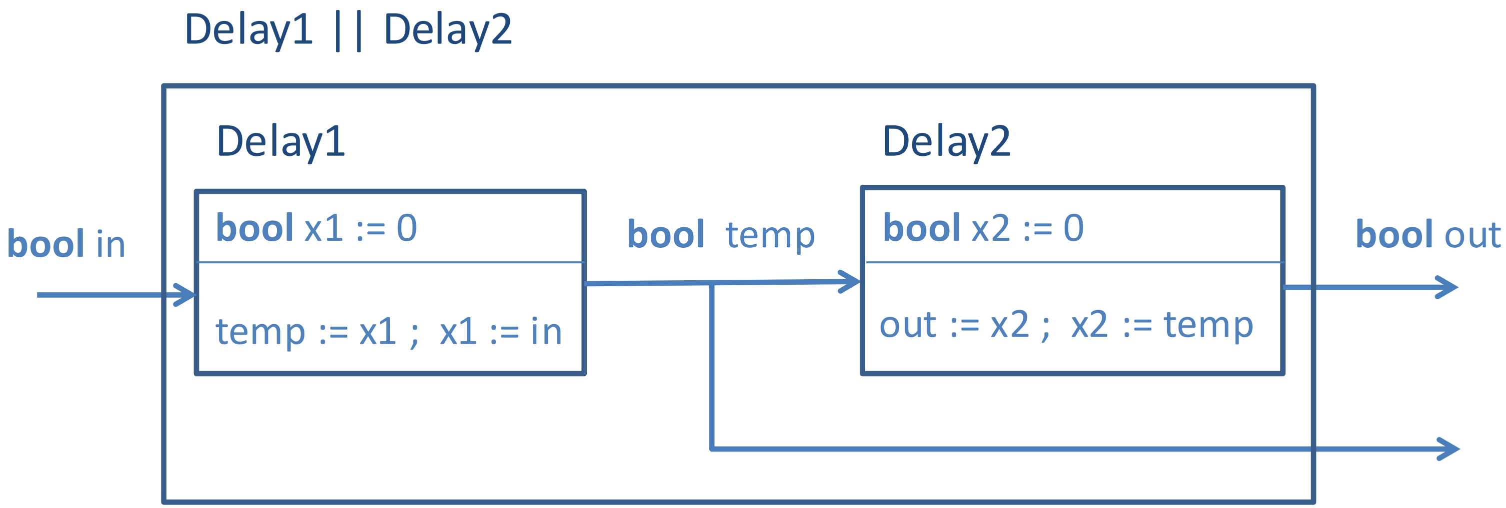
The state variables of Delay1 || Delay2 are { x1, x2 }

If C_1 has state vars S_1 and C_2 has state vars S_2

then $C_1 \,||\, C_2$ has state vars $S_1 \cup S_2$ (recall that $S_1 \cap S_2 = \emptyset$)

- A state of the product is a pair (s_1, s_2), where s_1 is a state of C_1 and s_2 is a state of C_2
- If C_1 has n_1 states and C_2 has n_2 states, then $C_1 \,||\, C_2$ has $n_1 \cdot n_2$ states

Initial States of Product



The initialization code Init for Delay1 || Delay2 is x1 := 0 ; x2 := 0

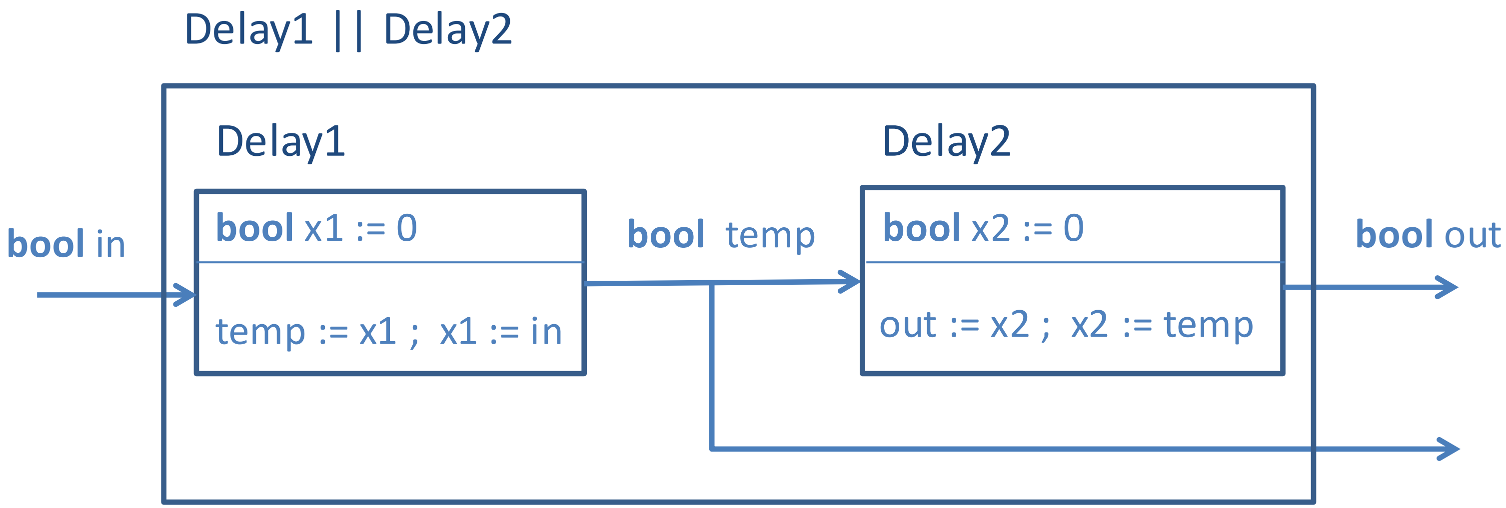
- Initial states are { (0,0) }

If C_1 has initialization Init_1 and C_2 has initialization Init_2

then $C_1 \,||\, C_2$ has initialization $\mathrm{Init}_1 \,;\, \mathrm{Init}_2$ (or, equivalently, $\mathrm{Init}_2 \,;\, \mathrm{Init}_1$)

- $[\![\mathrm{Init}]\!]$ is the Cartesian product $[\![\mathrm{Init}_1]\!] \times [\![\mathrm{Init}_2]\!]$

Reactions of Product

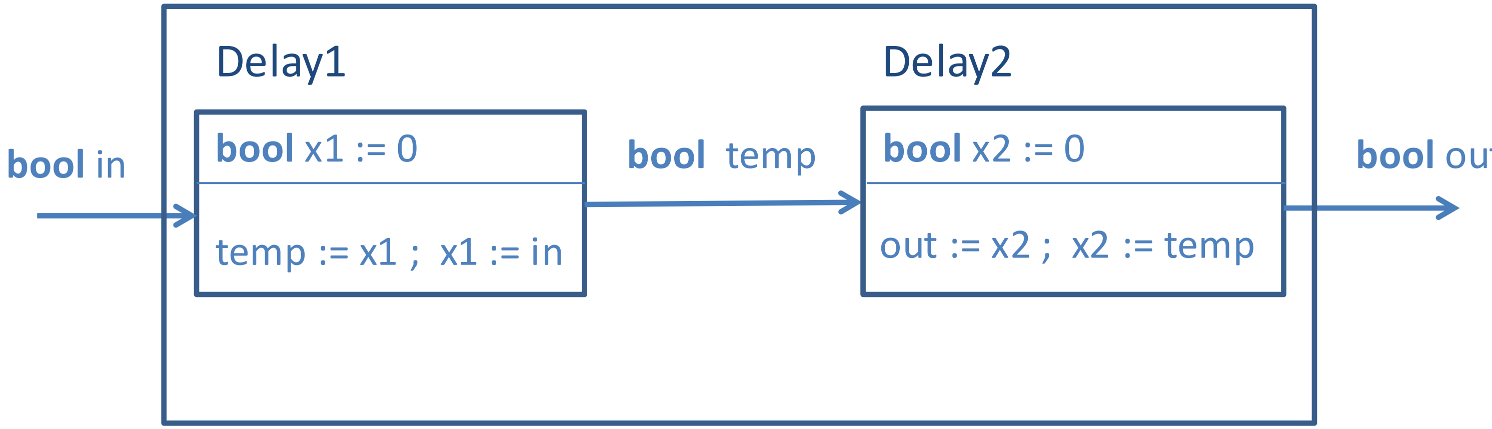


Execution of Delay1 || Delay2 within a round:

- Environment provides input value for variable in
- Concurrently execute code temp := x1 ; x1 := in of Delay1
- Concurrently execute code out := x2 ; x2 := temp of Delay2

Final Composition

(Delay[out ↦ temp] | | Delay[in ↦ temp]) \ temp

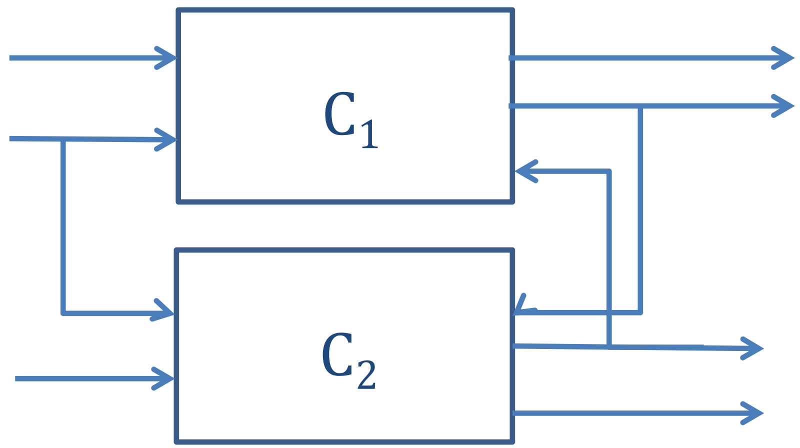


Instantiation: Delay[out ↦ temp] and Delay[in ↦ temp]

Parallel composition: Delay[out ↦ temp] | | Delay[in ↦ temp]

Encapsulation: (Delay[out ↦ temp] | | Delay[in ↦ temp]) \ temp

Feedback Composition



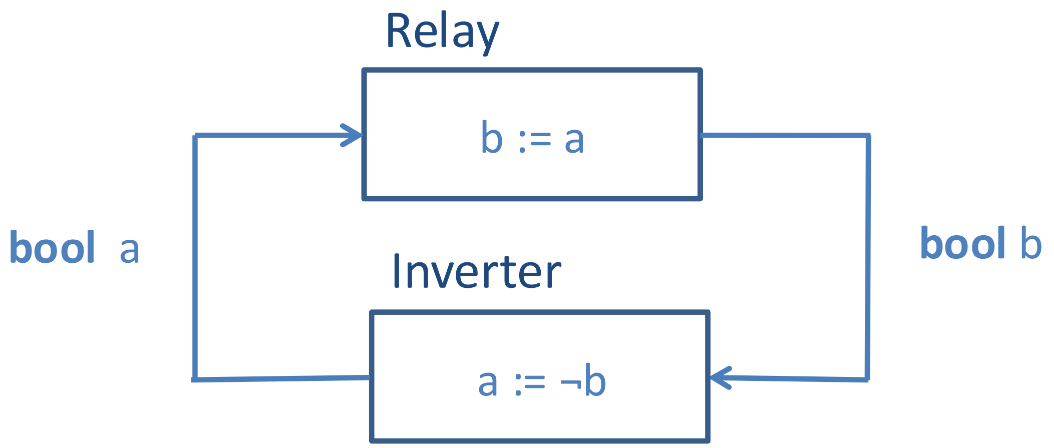
When

- some output of C_1 is an input of C_2, and
- some output of C_2 is an input of C_1,

how do we order the executions of their reactions React_1 and React_2?

Should such composition be allowed at all?

Feedback Composition

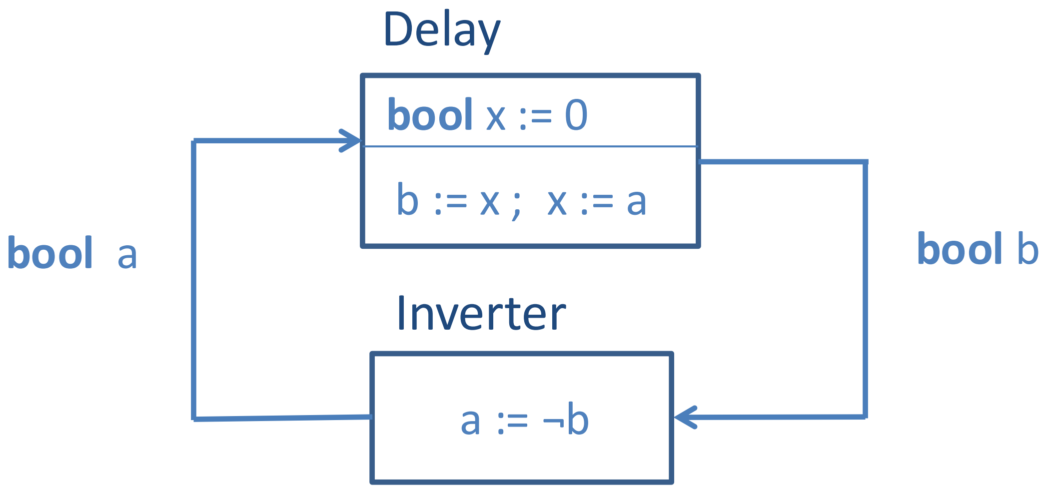


For Relay: its output b *awaits* its input a

For Inverter: its output a *awaits* its input b

- In product, we cannot order the execution of the two
- In the presence of such cyclic dependency, composition is disallowed
- **Intuition:** combinational cycles should be avoided

Feedback Composition



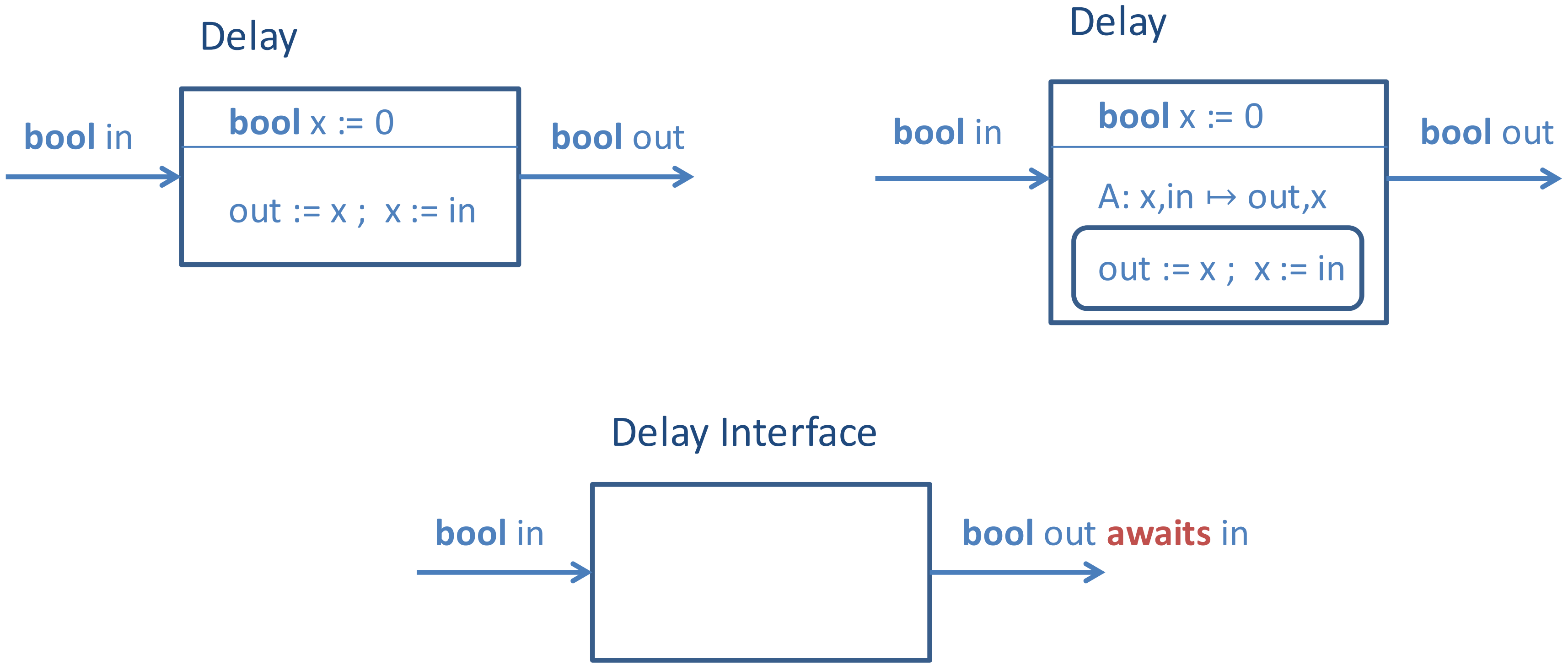
For Delay, output can be produced without waiting for input
(by executing the assignment b := x)

Reaction code for Delay || Inverter could be b := x ; a := ¬b ; x := a

Goal: Refine specification of reaction description so that
await dependencies among output-input vars are easy to detect

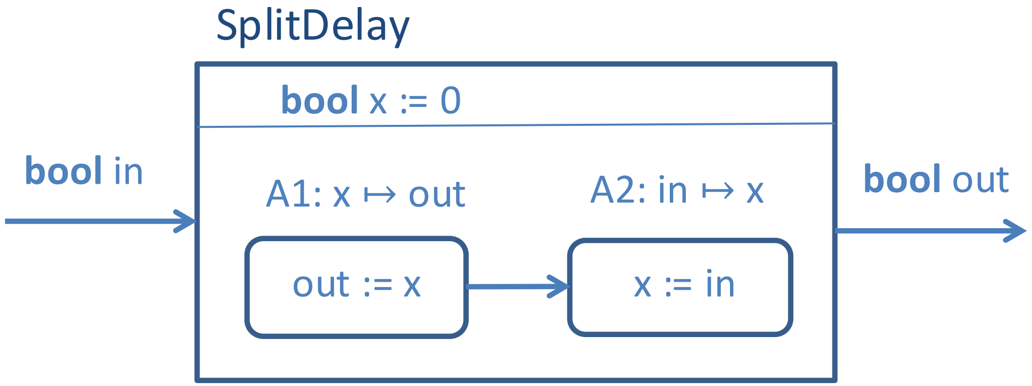
- Ordering of code-blocks during composition should be easy

Interfaces



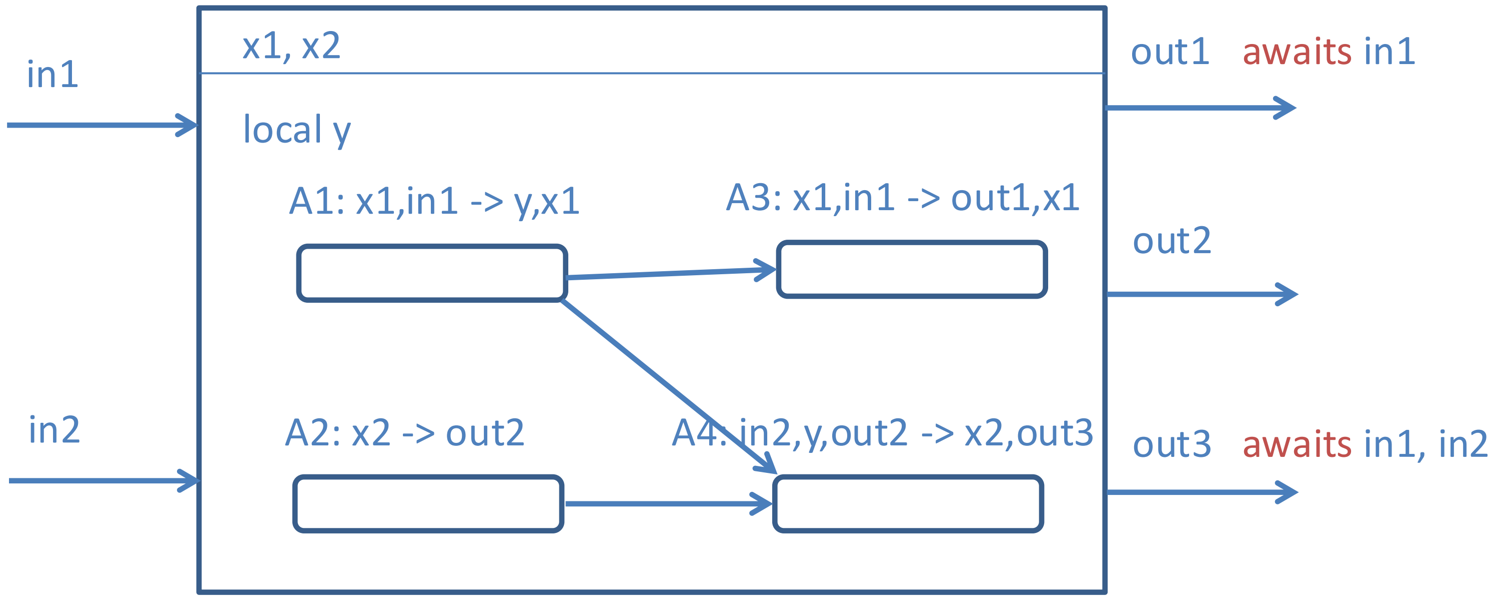
Interface = (input variables, output variables, await dependencies)

Interface: SplitDelay



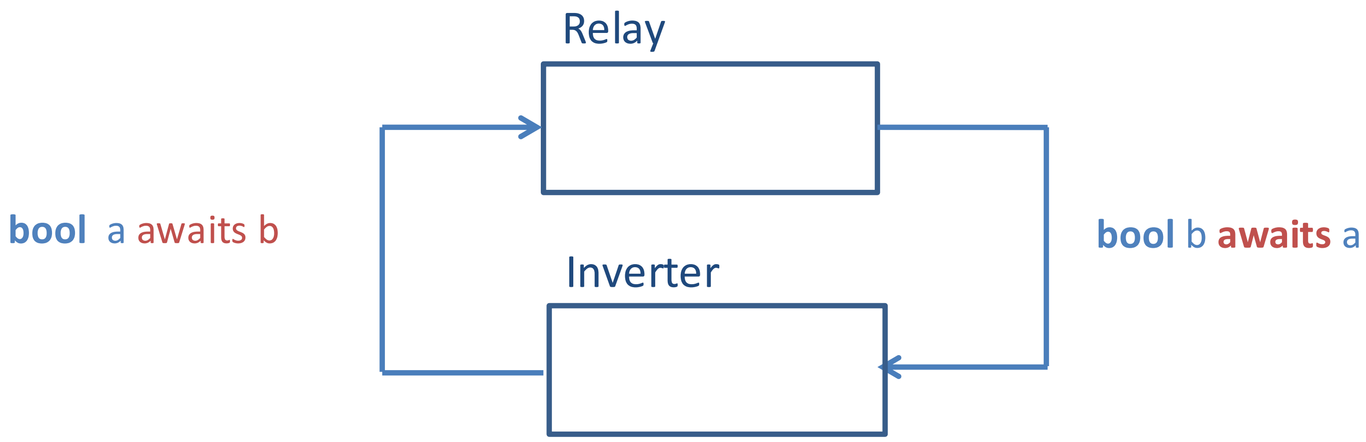
Decomposing the reaction into tasks eliminates in this case the await dependency between out and in

Example Interface



Example Interface

Back to Parallel Composition

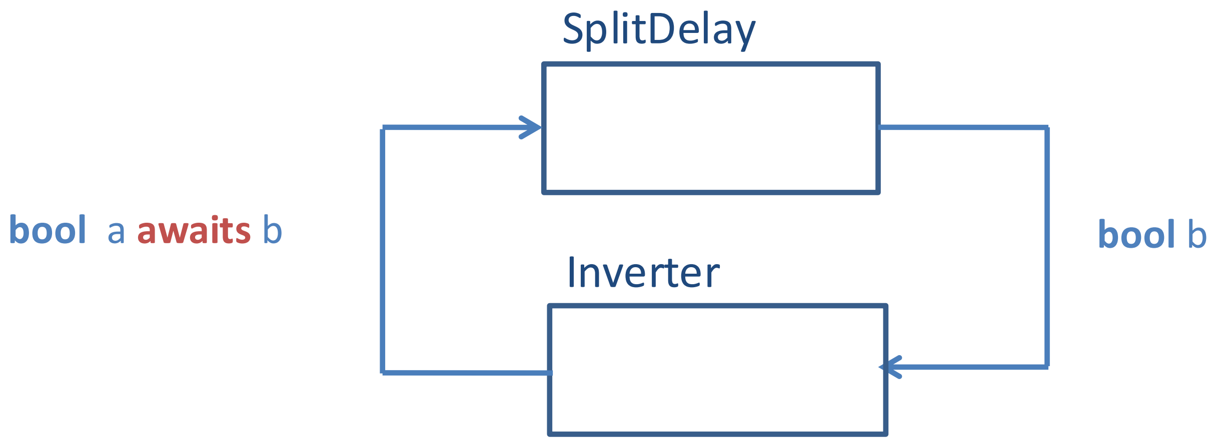


Relay and Inverter are not compatible

since there is a cycle in their combined await dependencies

Note: We can conclude that based just on the components' interfaces

Composing SplitDelay and Inverter



SplitDelay and Inverter are compatible

since there is no cycle in their combined await dependencies

Component Compatibility Definition

Given components :

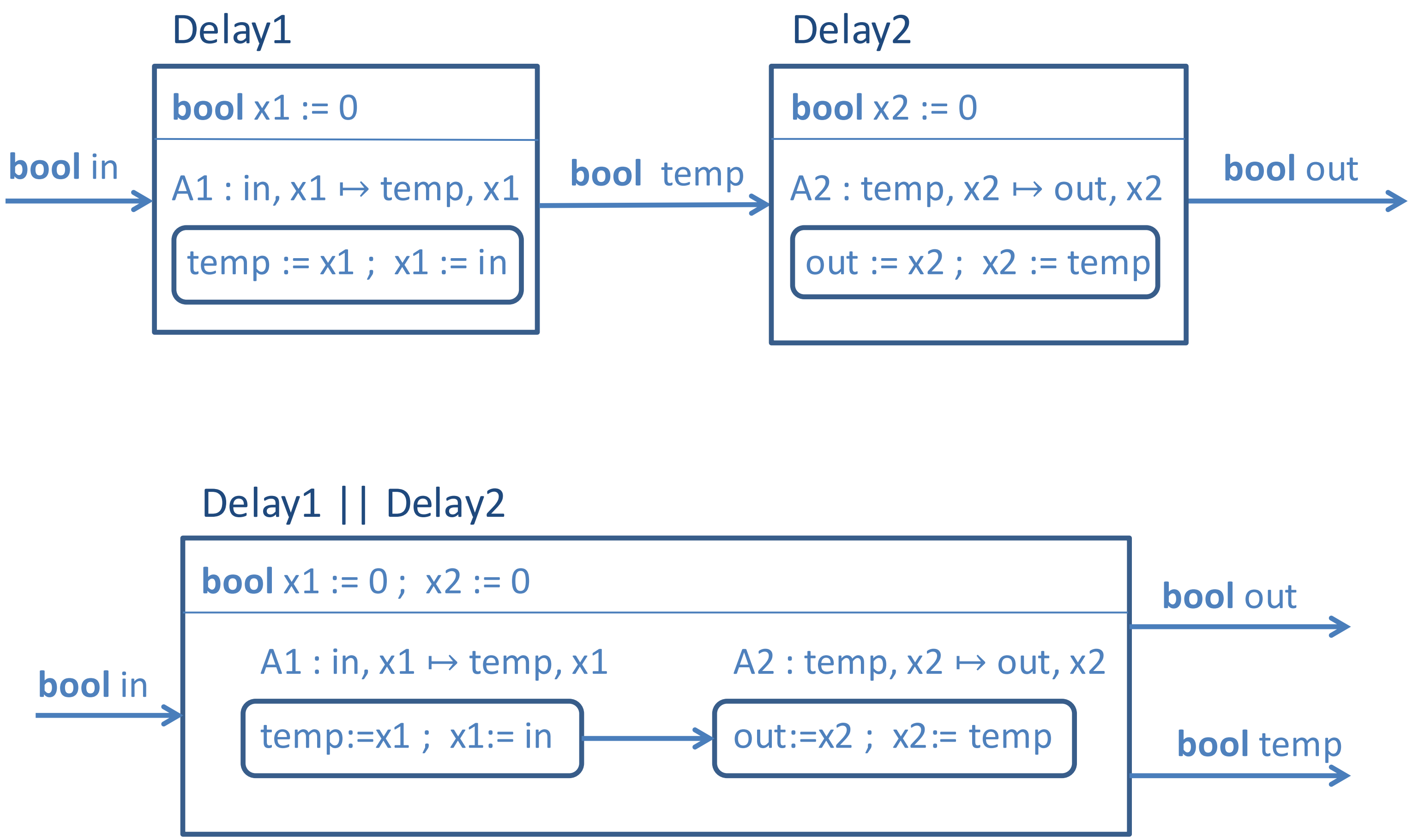
- C_1 with input vars I_1, output vars O_1, and awaits-dep. relation $\succ_1$
- C_2 with input vars I_2, output vars O_2, and awaits-dep. relation $\succ_2$

C_1 and C_2 are *compatible* if

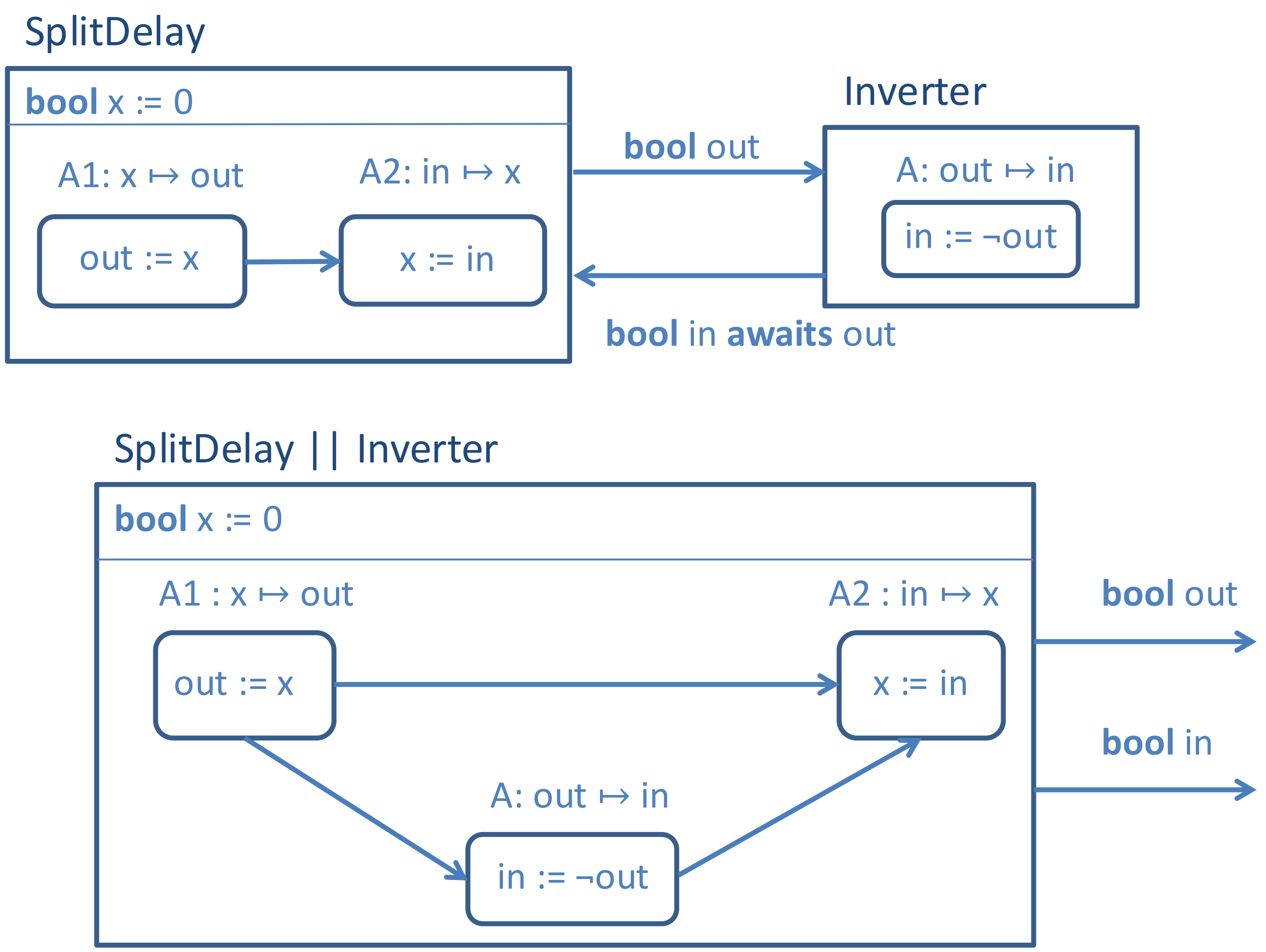
1. they have no common outputs: sets O_1 and O_2 are disjoint
2. the relation $\succ_1 \cup \succ_2$ of combined await-dependencies is acyclic

$$\boxed{\text{Parallel composition is allowed only for compatible components}}$$

Defining the Product



Composing SplitDelay and Inverter



Parallel Composition Definition

Given compatible components

- $C_1 = (I_1, O_1, S_1, Init_1, React_1)$ and
- $C_2 = (I_2, O_2, S_2, Init_2, React_2)$,

what's the reaction of product $C = C_1 \mid\mid C_2$?

Suppose $React_1$ and $React_2$ are specified using resp.

- local vars L_1, set of tasks P_1, and precedence $\prec_1$, and
- local vars L_2, set of tasks P_2, and precedence $\prec_2$

Reaction description for product C has

- local variables $L_1 \cup L_2$
- set of tasks $P_1 \cup P_2$
- precedence edges $\prec_1 \cup \prec_2 \cup \{$ edges between task A_1 of C_i and A_2 of C_j $\mid$ $i \neq j$ and A_2 reads a var written by $A_1\}$

Parallel Composition Definition

Why is the parallel composition operation well-defined?

- Can the new edges make task graph of the product cyclic?

Recall: Await-dependencies among I/O variables of compatible components must be acyclic

Proposition 2.1: Awaits compatibility implies acyclicity of product task graph

Bottom line: Interfaces capture enough information to define parallel composition in a consistent manner

Aside: It is possible to define more flexible (but more complex) notions of awaits dependency

Properties of Parallel Composition

Commutativity: $C_1 \,||\, C_2 \;=\; C_2 \,||\, C_1$ (when C_1, C_2 are compatible)

Associativity: $(C_1 \,||\, C_2) \,||\, C_3 \;=\; C_1 \,||\, (C_2 \,||\, C_3)$

- If compatibility check fails in one case, it will also fail in the others

Bottom line: order of composition does not matter

Note:

- If C_1 has n_1 states and C_2 has n_2 states, then $C_1 \,||\, C_2$ has $n_1 \cdot n_2$ states
- If both C_1 and C_2 are deterministic, so is $C_1 \,||\, C_2$
- If both C_1 and C_2 are event-triggered, is $C_1 \,||\, C_2$ guaranteed to be event-triggered?

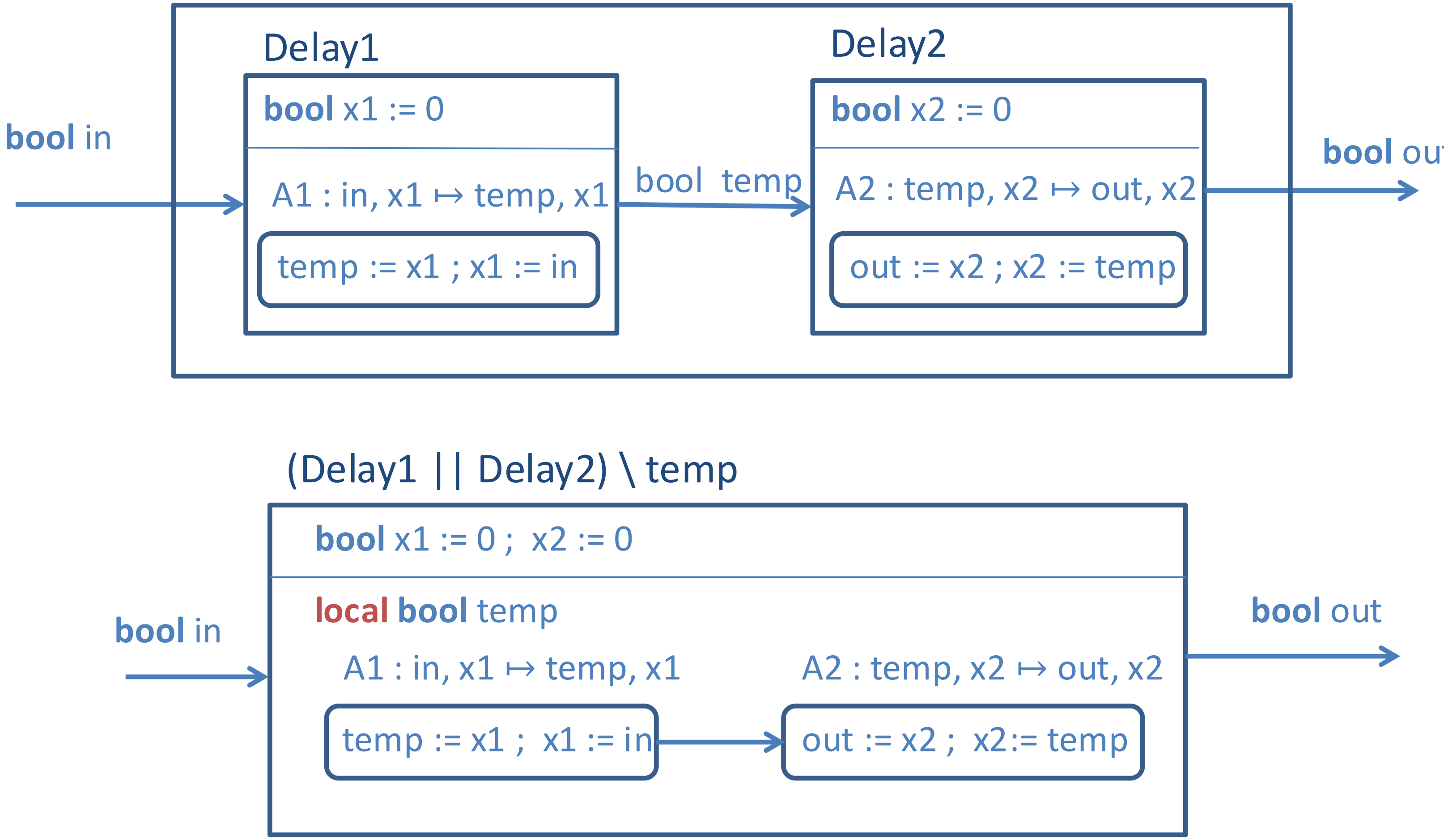
Output Hiding

Let C be a component and y one of its output vars

- The result of hiding y in C, written as C \ y, is a component identical to C except that y is a local variable

This is useful for limiting the scope or a component (encapsulation)

DoubleDelay



Credits

Notes based on Chapter 2 of

Principles of Cyber-Physical Systems
by Rajeev Alur
MIT Press, 2015