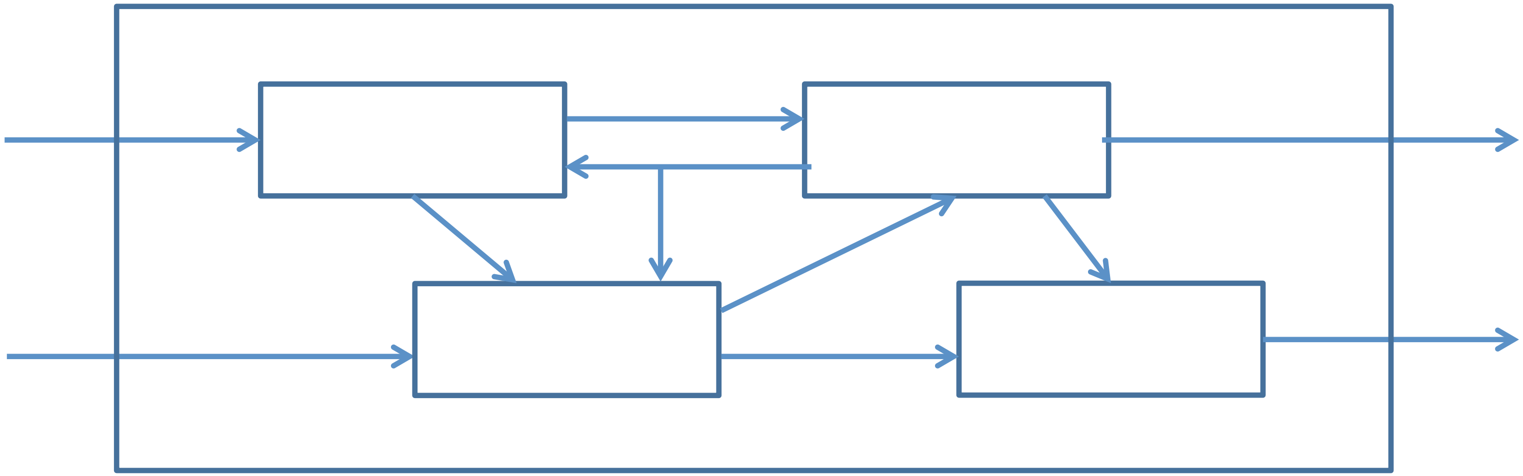
CS:4980
Foundations of Embedded Systems

Synchronous Model
Part II

Copyright 20014-16, 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.

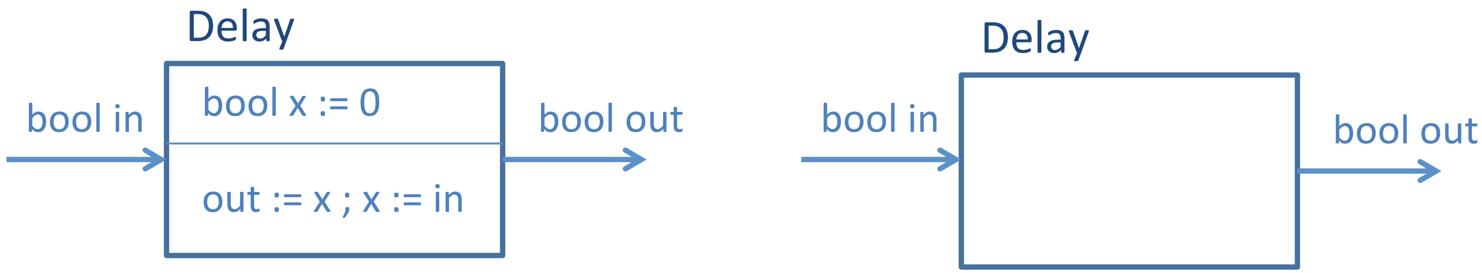
Block Diagrams



Structured modeling

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

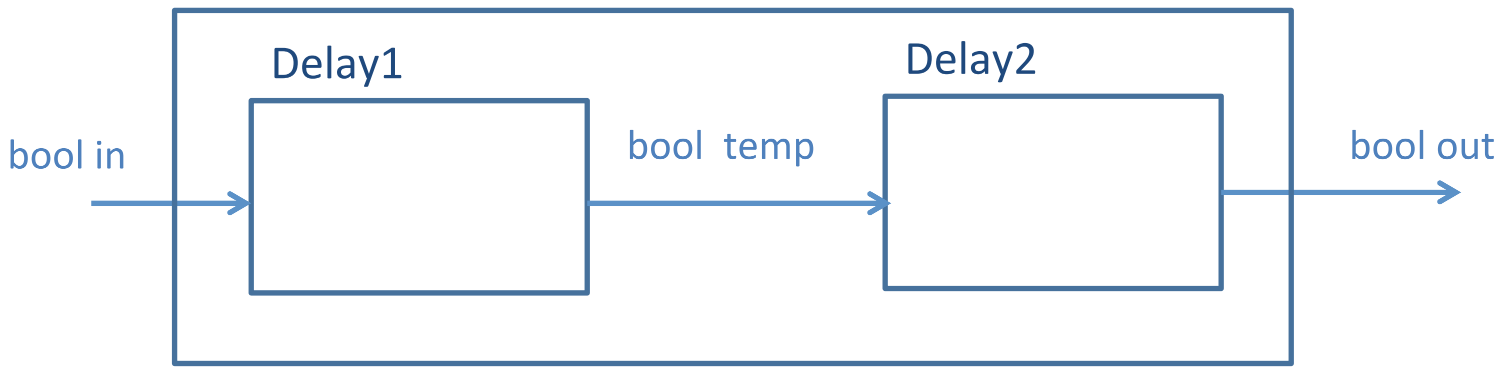
DoubleDelay



Design a component with

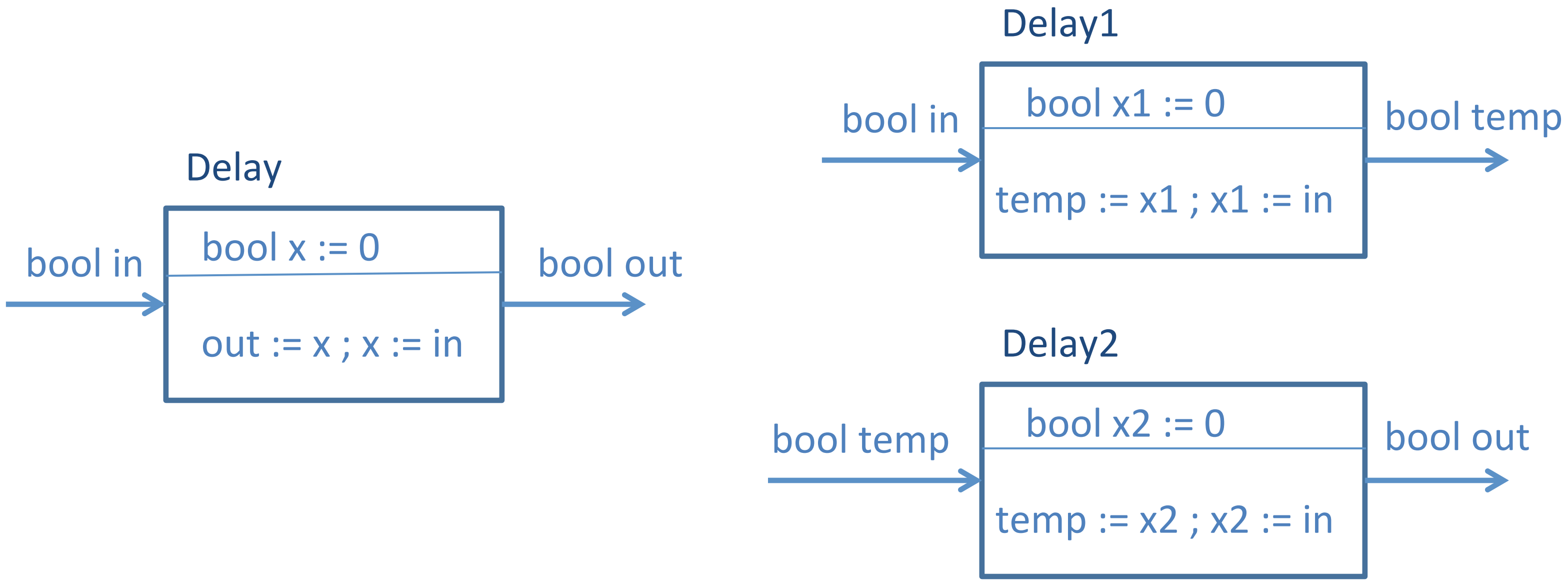
- Input: bool in
- Output: bool out
- Output in round n should equal input in round n-2

DoubleDelay



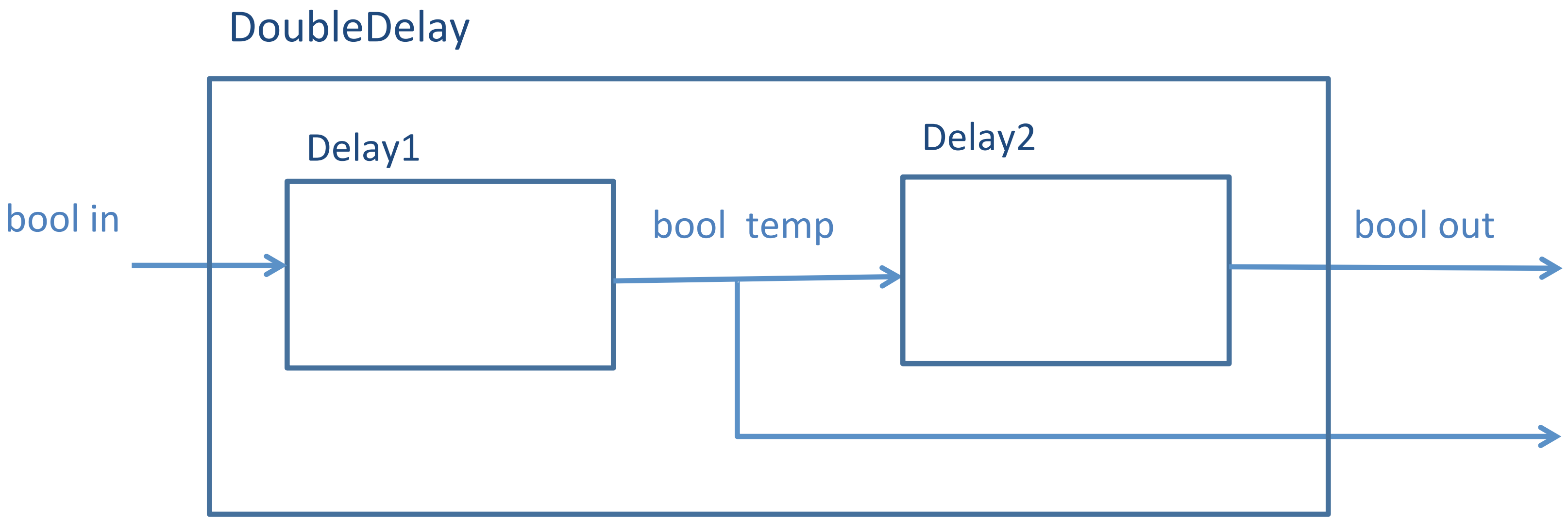
- Instantiation: Create two instances of Delay
  - Output of Delay1 = Input of Delay2 = Variable temp
- Parallel composition: Concurrent execution of Delay1 and Delay2
- Encapsulation/Hiding: Hide variable temp

Instantiation / Renaming



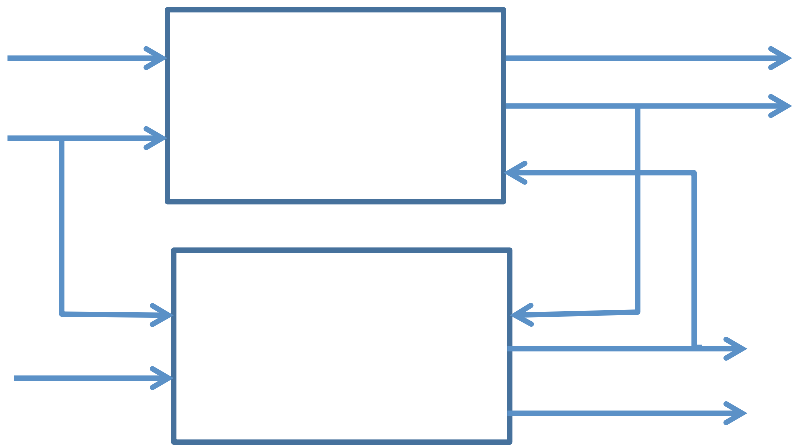
- Delay1 = Delay[out ↦ temp]
  - Explicit renaming of input/output variables
  - Implicit renaming of state variables
  - Components (I, O, S, Init, React) of Delay1 derived from Delay
- Delay2 = Delay[in ↦ temp]

Parallel Composition



- DoubleDelay = Delay1 || Delay2
  - Execute both 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



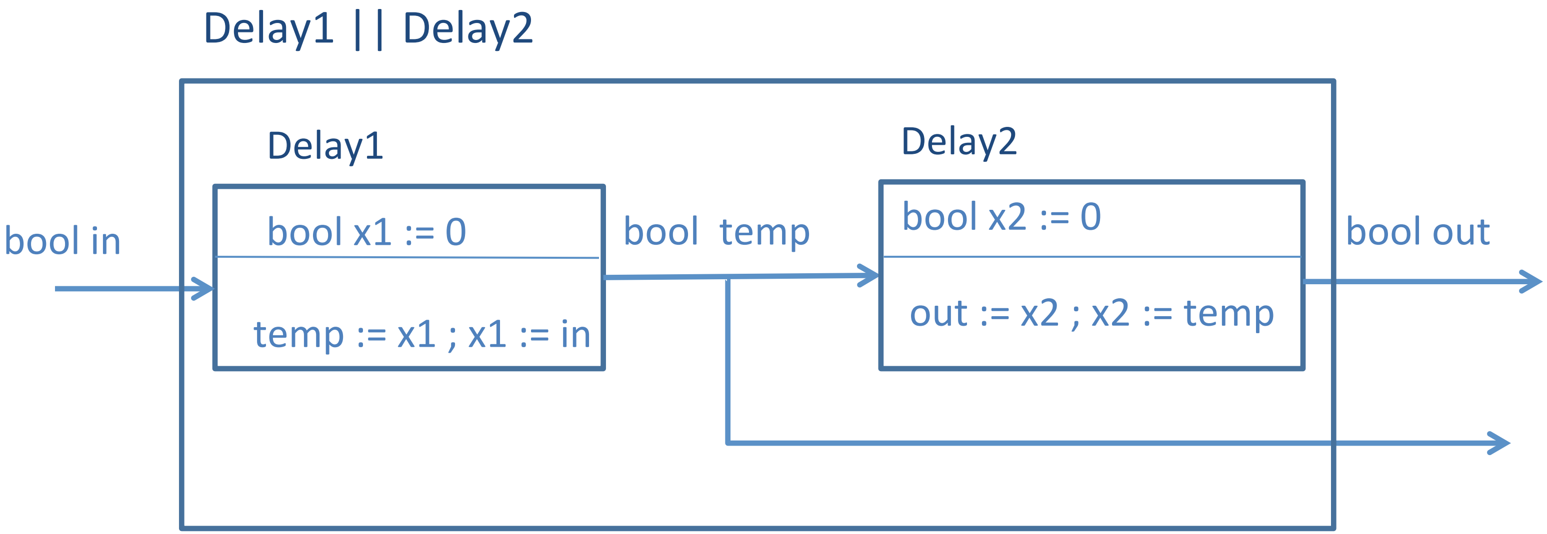
Allowed:

- input variables in common
- output variable of one is input variable of the other

Disallowed:

- common output variables
  - a unique component must be responsible for values of any given variable
- common state variables
  - state variables can be implicitly renamed to avoid conflicts

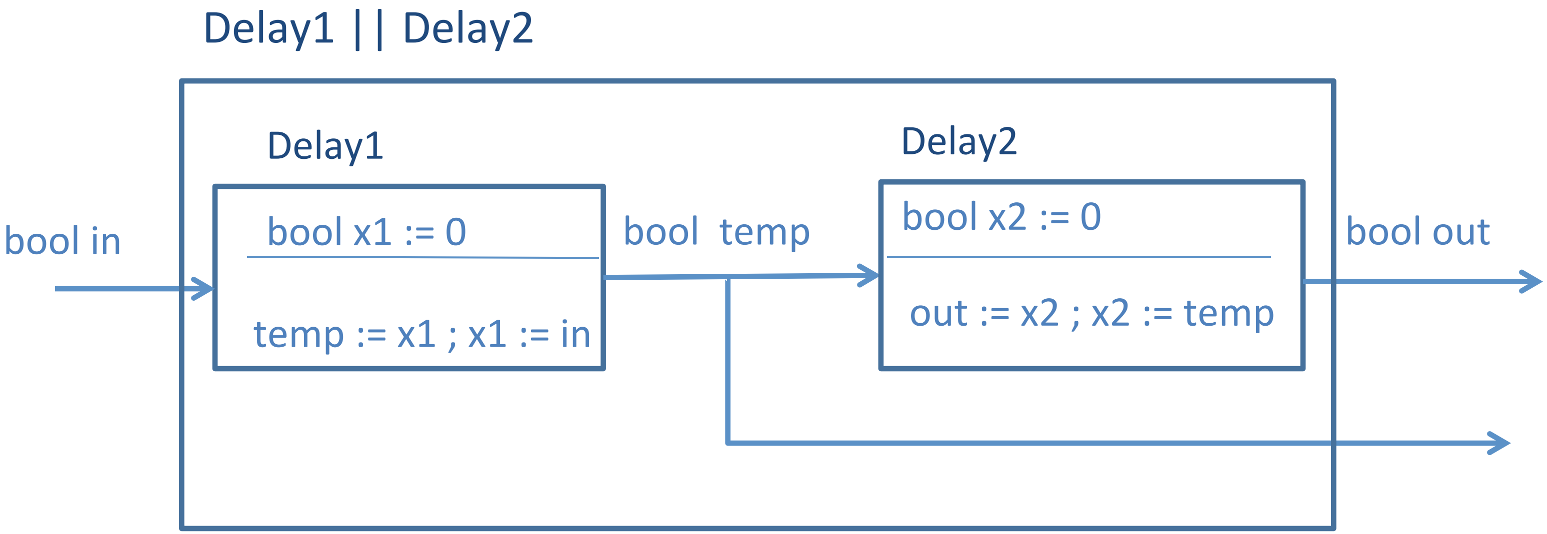
Outputs of Product



- ❑ Output variables of Delay1 || Delay2 are {temp, out}
  - Note: by default, every output is available to outside world

- ❑ If C1 has output vars O1 and C2 has output vars O2 then the product C1 || C2 has output vars O1 ∪ O2

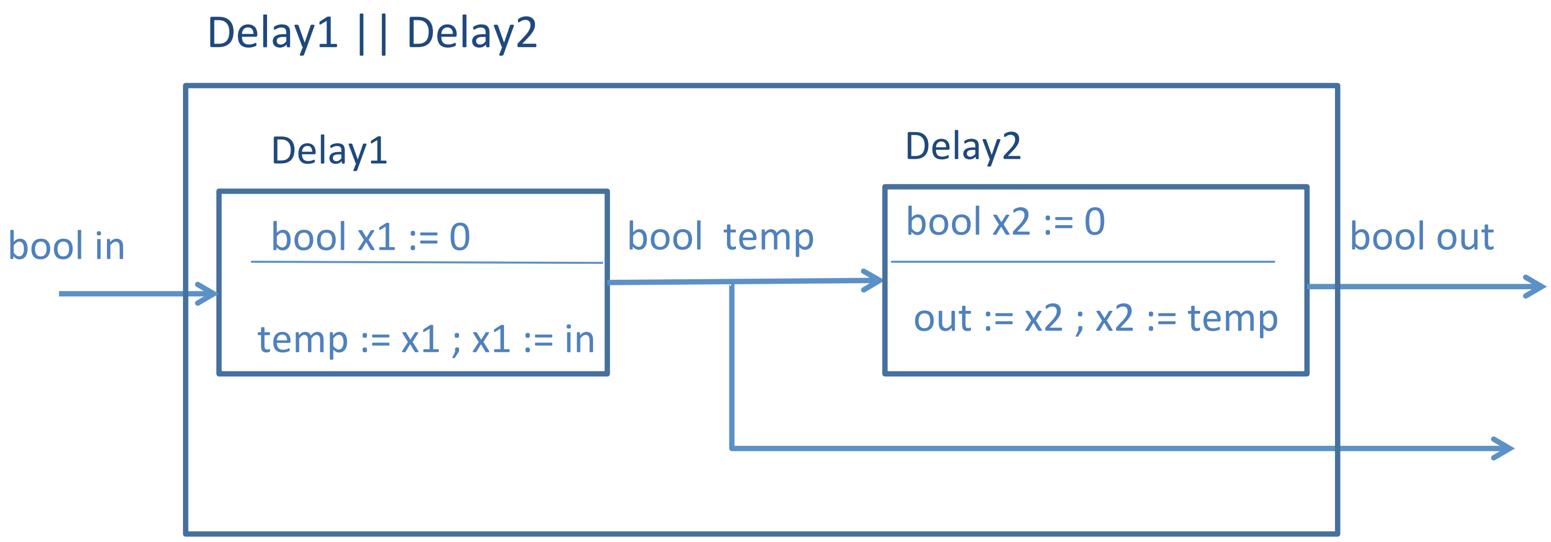
Inputs of Product



- Input variables of Delay1 || Delay2 are {in}
  - Even though temp is input of Delay2, it is not an input of product

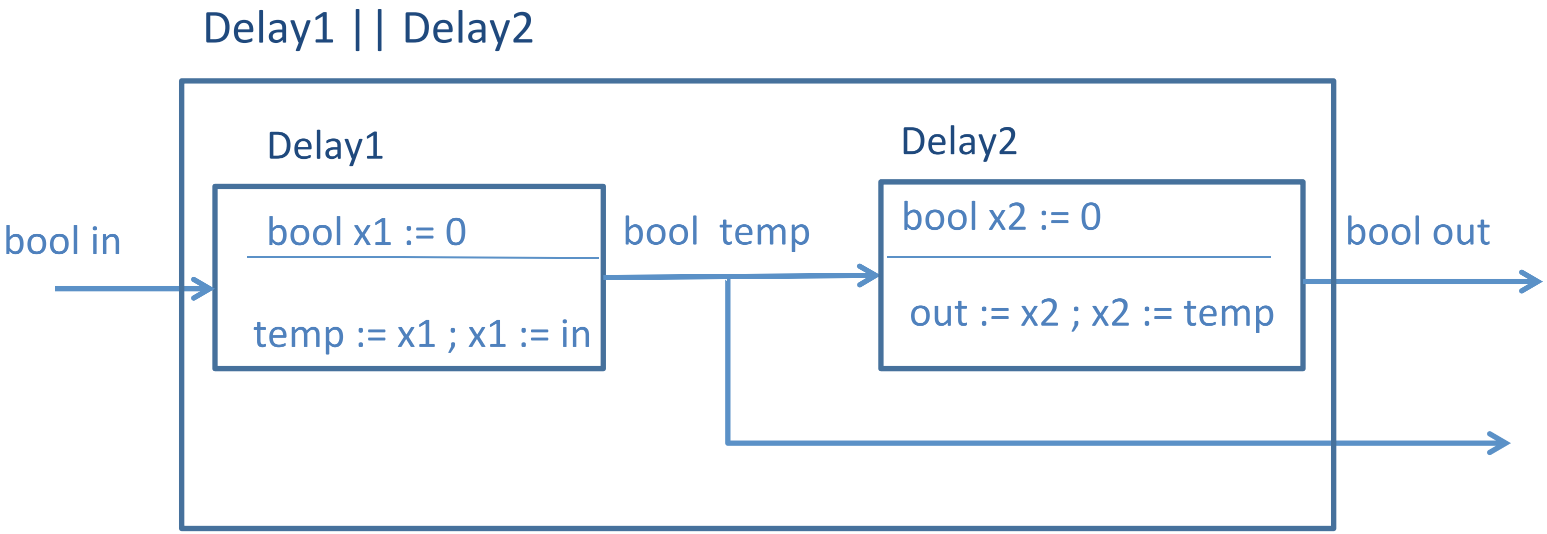
- If C1 has input vars I1 and C2 has input vars I2 then C1 || C2 has input vars (I1 ∪ I2) \ (O1 ∪ O2)
  - A variable is an input of the product if it is an input of one of the components, and not an output of the other

States of Product



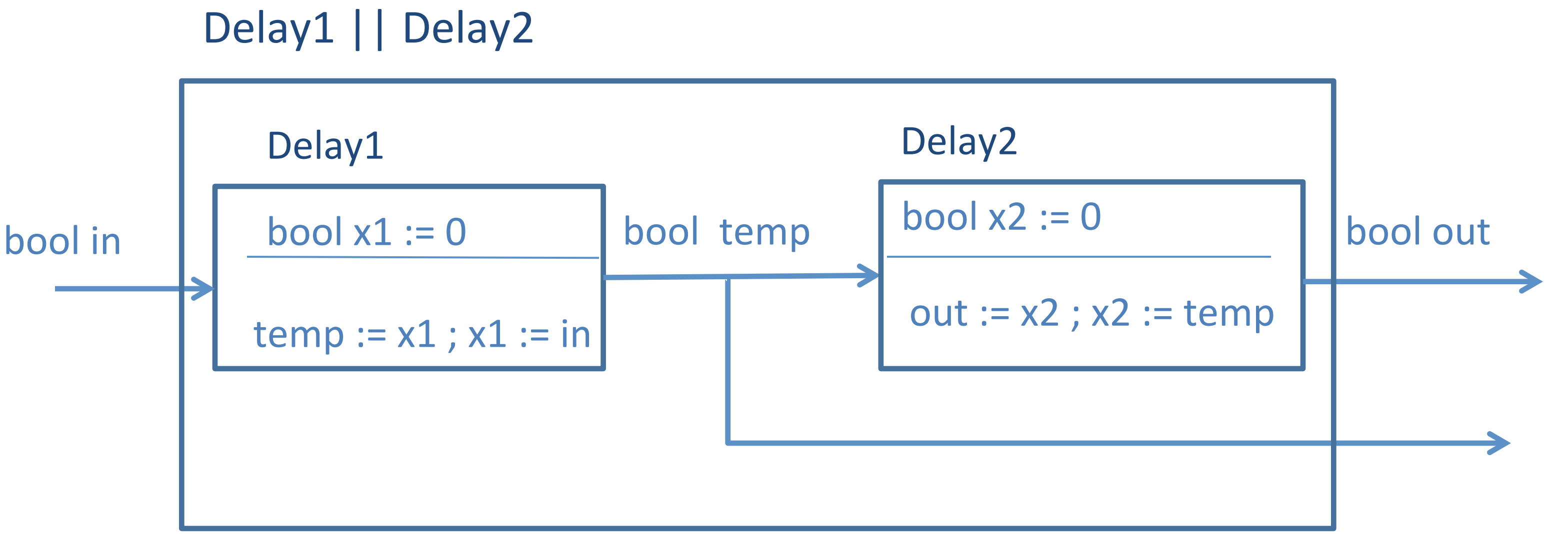
- State variables of Delay1 || Delay2 are {x1, x2}
- If C_1 has state vars S_1 and C_2 has state vars S_2 then $C_1 \| C_2$ has state vars $S_1 \cup S_2$ (recall that $S_1 \cap S_2 = \emptyset$)
  - A state of the product is a pair (s_1, s_2), where s_1 is a state of C_1 and s_2 is a state of C_2
  - 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 $Init_1$; $Init_2$ (or $Init_2$; $Init_1$)
- Order does not matter
  - [Init] is the Cartesian product $[Init_1] \times [Init_2]$

Reactions of Product

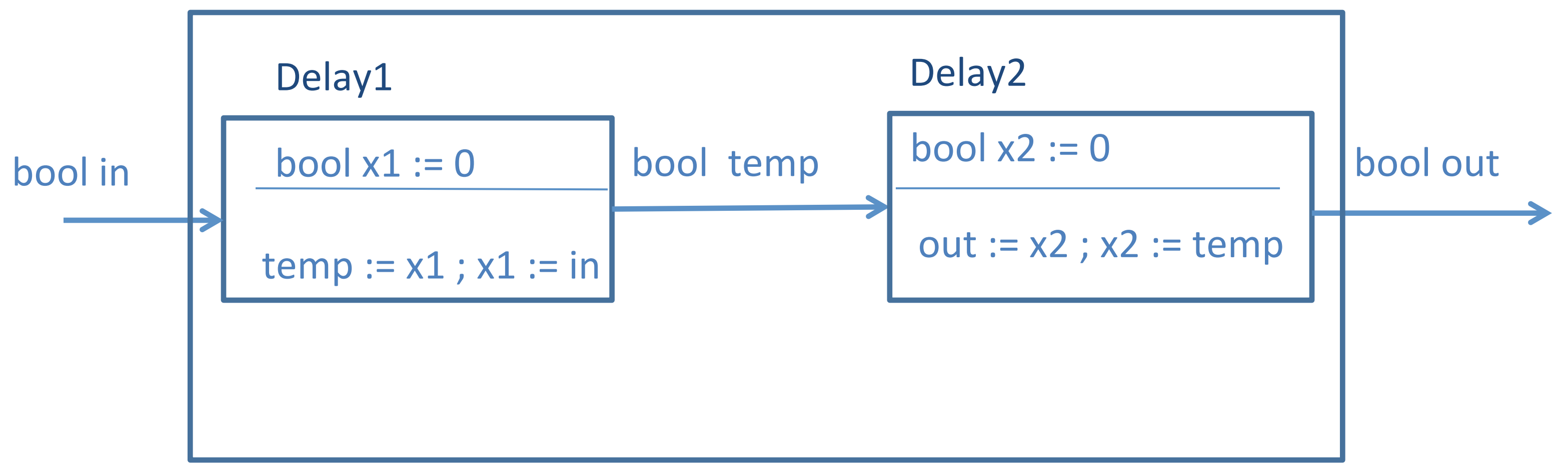


Execution of Delay1 || Delay2 within a round

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

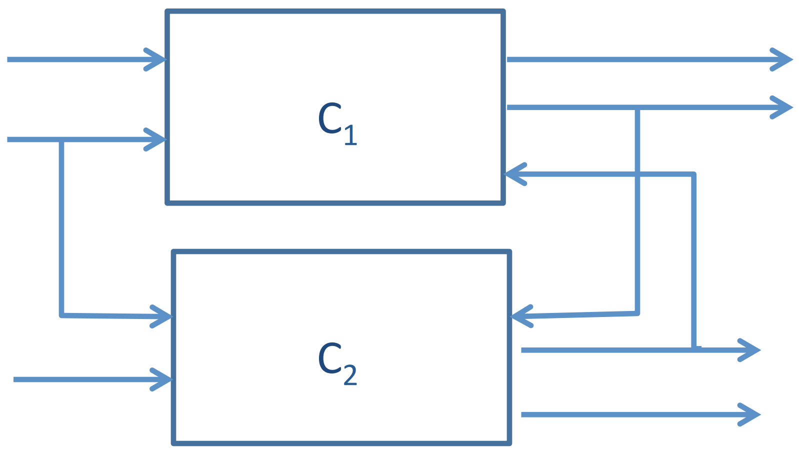
Final Composition

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



- Instantiation: Delay[out ↦ temp] and Delay[in ↦ temp]
- Parallel composition: Delay[out ↦ temp] || Delay[in ↦ temp]
- Output hiding: (Delay[out ↦ temp] || Delay[in ↦ temp]) \ temp

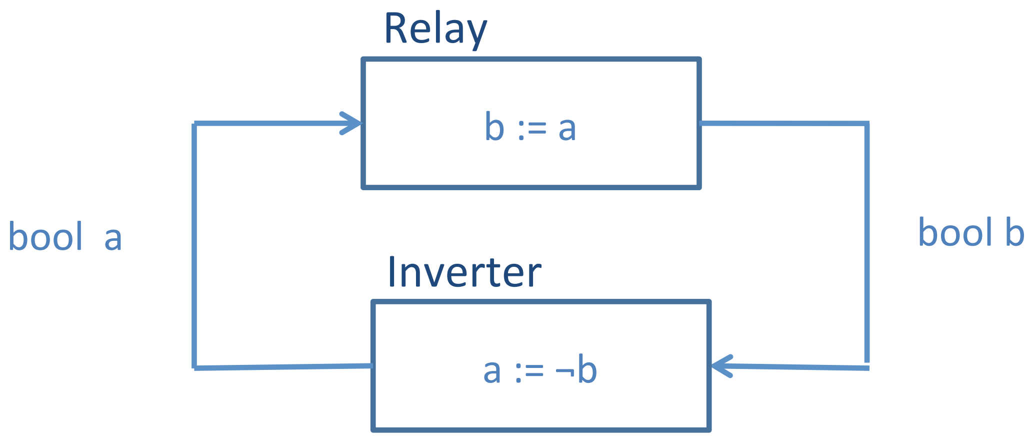
Feedback Composition



- 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 reaction $React_1$ and $React_2$?
- Should such composition be allowed at all?

Feedback Composition

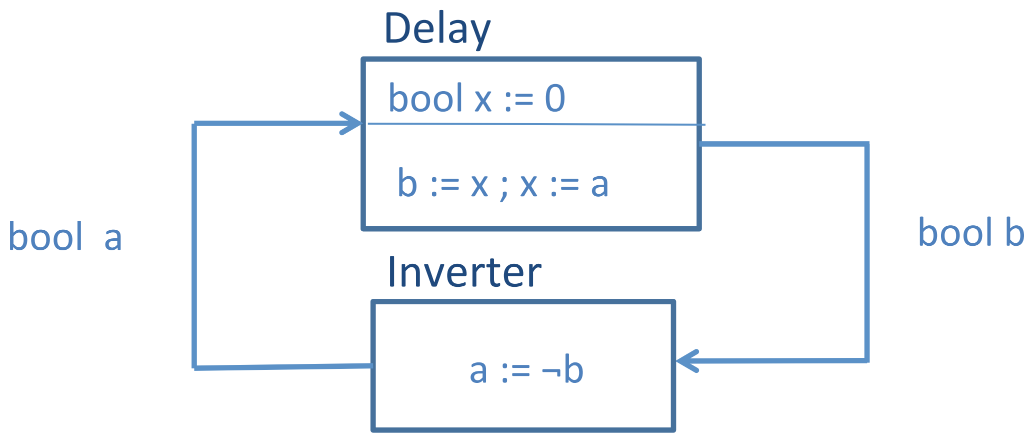


For Relay: its output b awaits its input a

For Inverter: its output a awaits its input b

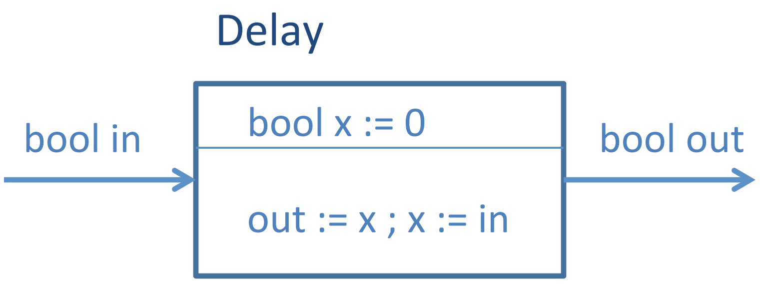
- In product, cannot order the execution of the two
- In the presence of such cyclic dependency, composition is disallowed
- Intuition: Combinational cycles should be avoided

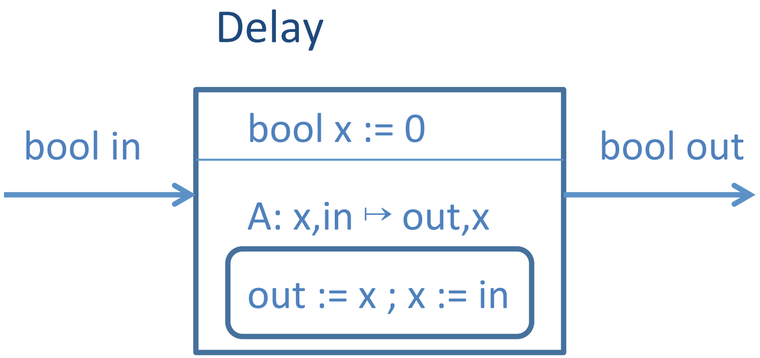
Feedback Composition

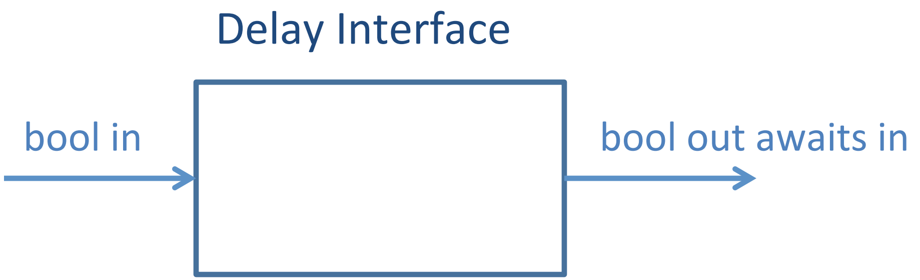


- For Delay, possible to produce output without waiting for its input by executing the assignment b := x
- Reaction code for Delay || Inverter could be b := x ; a := ¬b ; x := a
- Goal: Refine specification of reaction description so that await dependencies among output-input variables are easy to detect
  - Ordering of code-blocks during composition should be easy

Interfaces

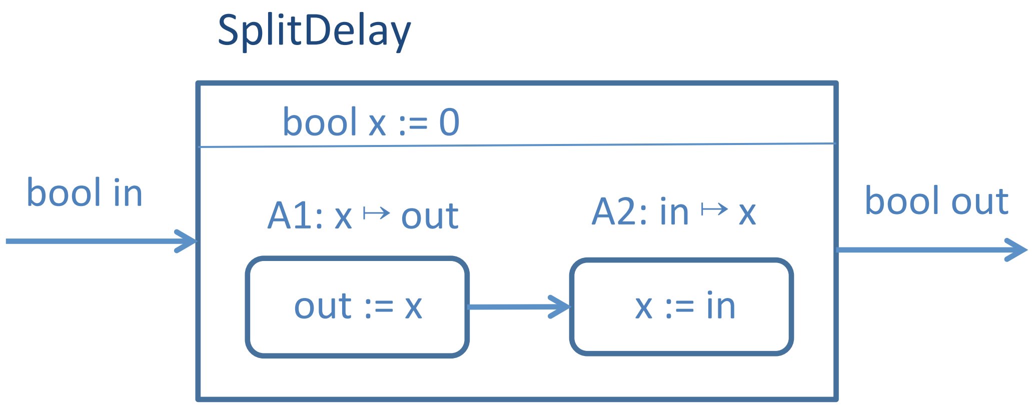






Interface = (input variables, output variables, await dependencies)

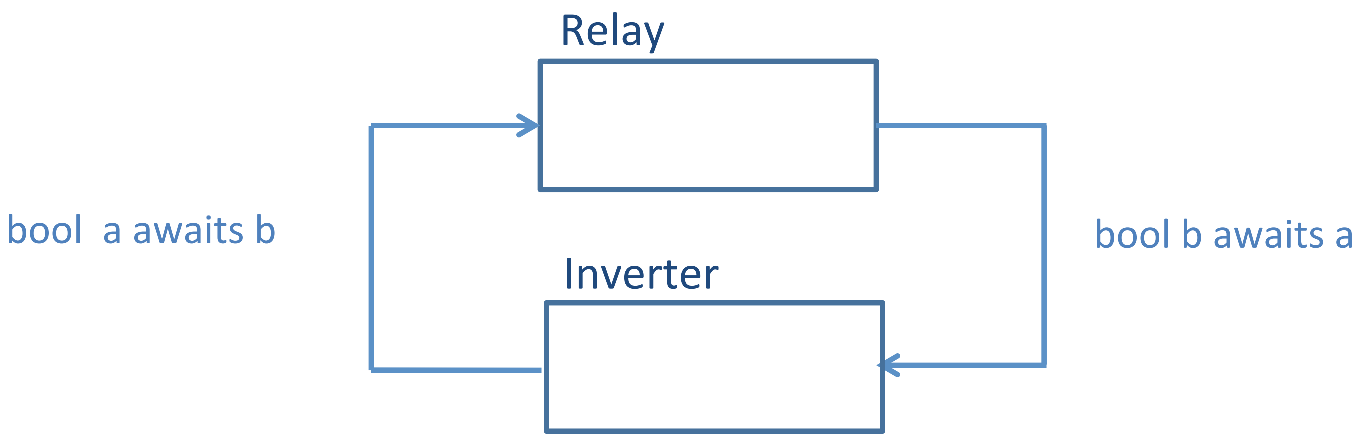
Interface: SplitDelay



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

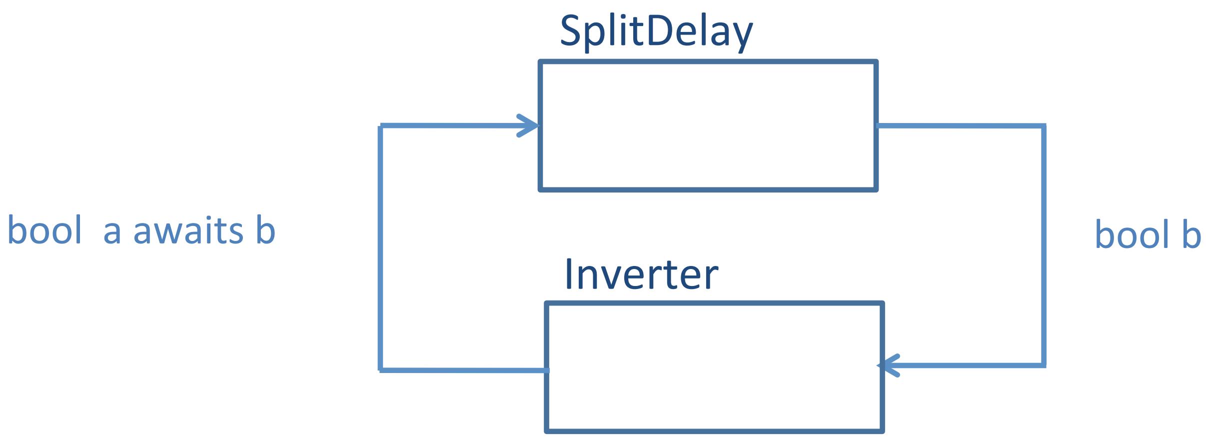
Example Interface

Back to Parallel Composition



Relay and Inverter are not compatible since there is a cycle in their combined await dependencies

Composing SplitDelay and Inverter



SplitDelay and Inverter are compatible since there is no cycle in their combined await dependencies

Note: Based on their interfaces, Delay and Inverter are not compatible

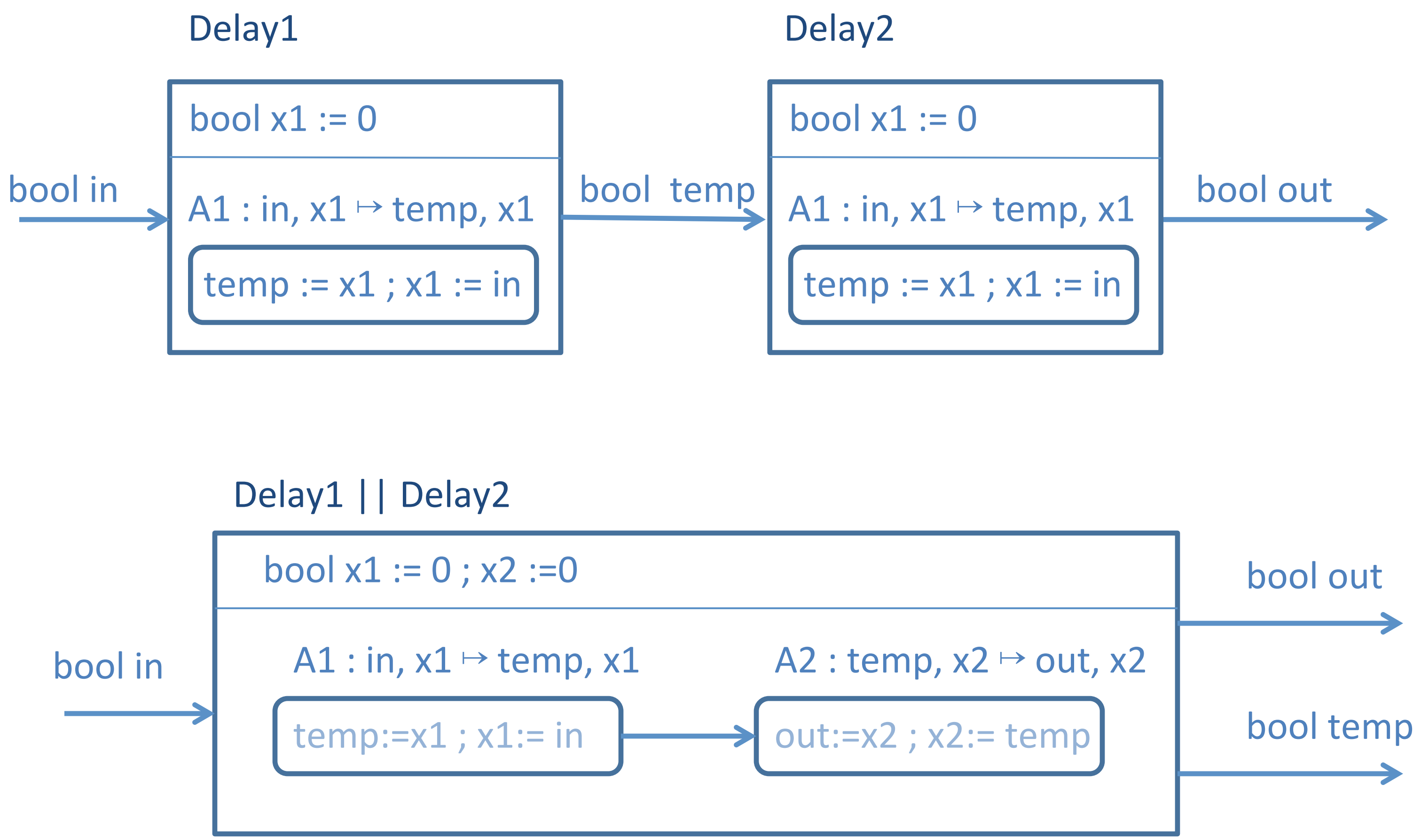
Component Compatibility Definition

- Given components :
  - C_1 with input vars I_1, output vars O_1, and awaits-dep. relation $>_1$
  - C_2 with input vars I_2, output vars O_2, and awaits-dep. relation $>_2$

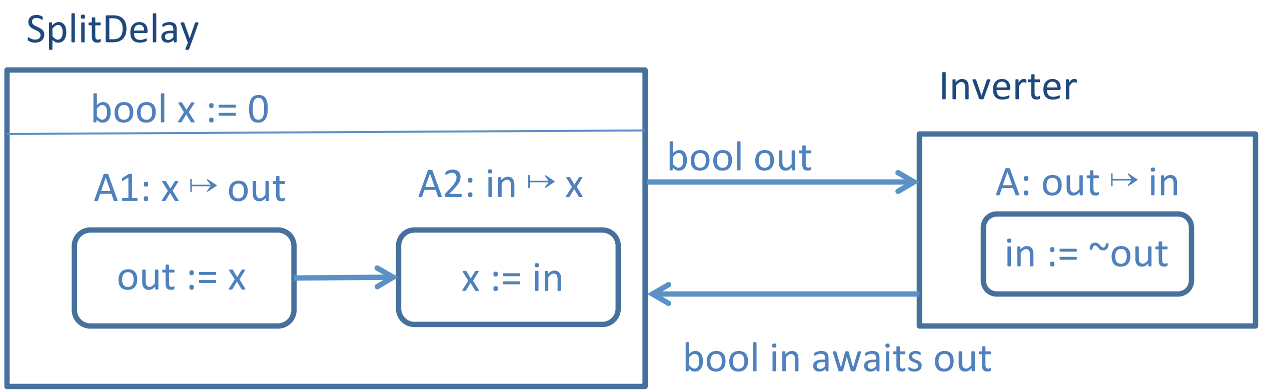
- C_1 and C_2 are compatible if
  - they have no common outputs: sets O_1 and O_2 are disjoint
  - the relation $>_1 \cup >_2$ of combined await-dependencies is acyclic

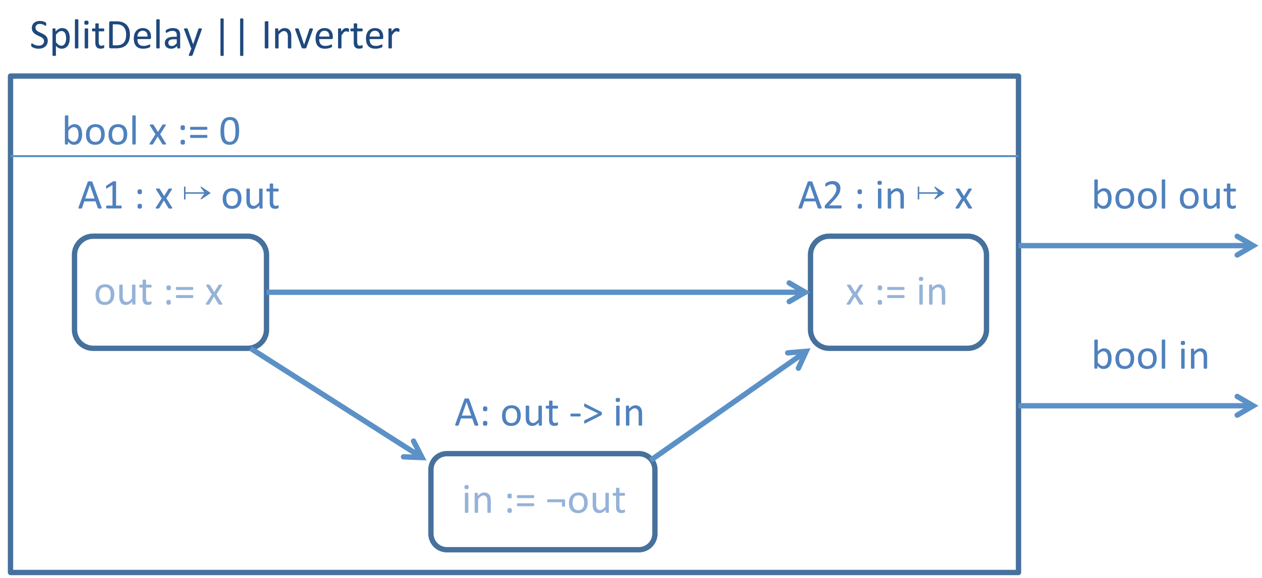
- Parallel Composition is allowed only for compatible components

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 definition of product $C = C_1 \,||\, C_2$?

- Suppose $React_1$ and $React_2$ are specified using resp.
  - local vars L_1, set of tasks P_1, and precedence $<_1$, and
  - local vars L_1, set of tasks P_2, and precedence $<_2$

- Reaction description for product C has
  - local variables $L_1 \cup L_2$
  - set of tasks $P_1 \cup P_2$
  - precedence edges $<_1 \cup <_2 \cup$ {edges between tasks A_1 and A_2 of different components if 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: possible to define more flexible (but more complex) notions of awaits dependencies

Properties of Parallel Composition

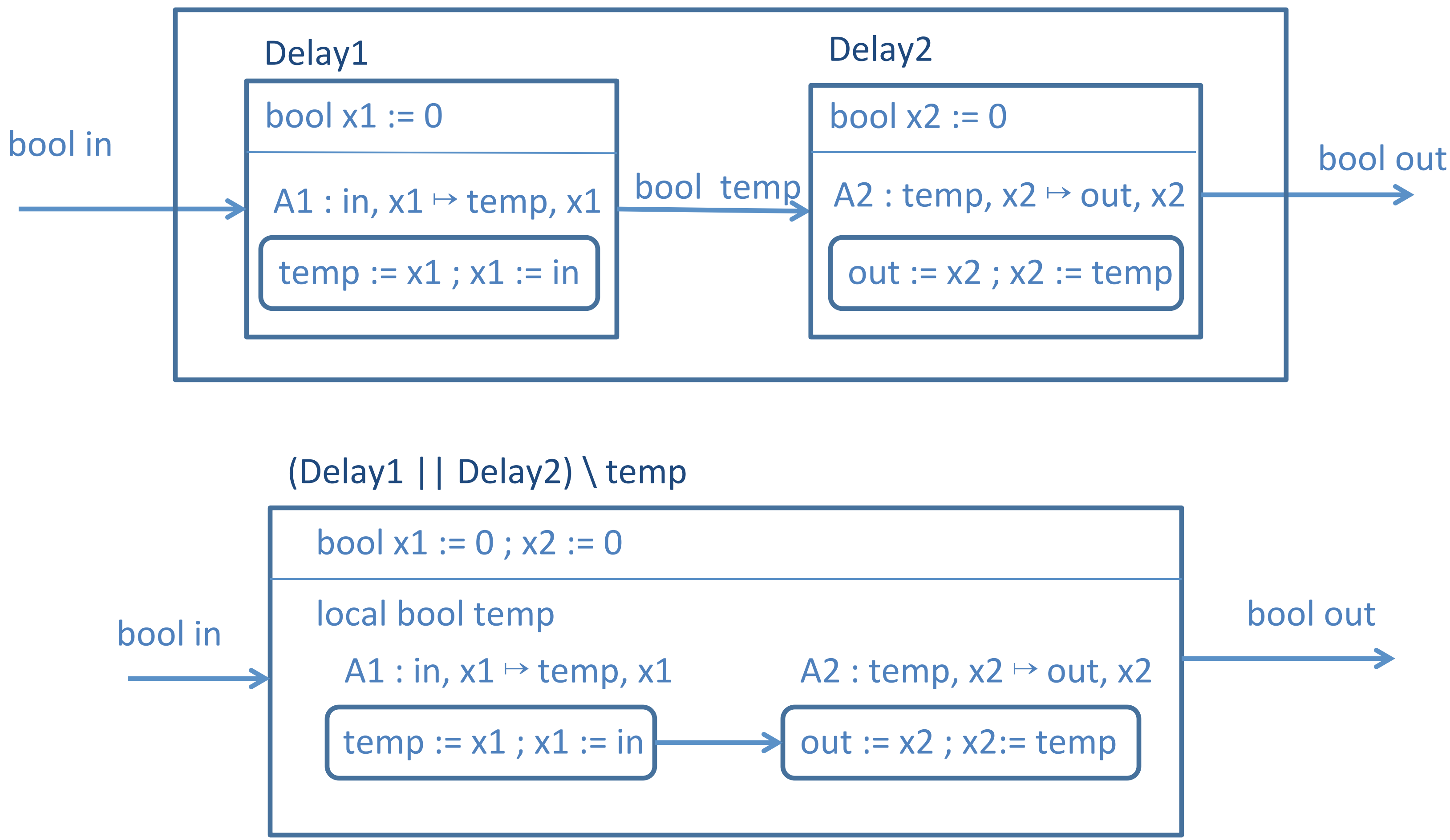
- Commutative: $C_1 \;||\; C_2 \;=\; C_2 \;||\; C_1$
- Associative: $(C_1 \;||\; C_2) \;||\; C_3 \;=\; C_1 \;||\; (C_2 \;||\; C_3)$
  - If compatibility check fails in one case, will also fail in others
- Bottom line: order of composition does not matter
- If C_1 has n_1 states and C_2 has n_2 states 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 no longer an output variable but 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