Yet Another Reo Semantics: Reasoning about Speculative Execution

Hans-Dieter A. Hiep

Vrije Universiteit Amsterdam (MSc student)
Centrum Wiskunde & Informatica (Intern)

September 25, 2018

Overview

- Master's thesis:
  Yet Another Reo Semantics: Reasoning about Speculative Execution

Overview

- Master's thesis:
  Yet Another Reo Semantics: Reasoning about Speculative Execution

- Outline:
  1. Background and Challenges

Overview

- Master's thesis:
  Yet Another Reo Semantics: Reasoning about Speculative Execution

- Outline:
  1. Background and Challenges
  2. Syntax
     typed language for compositions and components

Overview

- Master's thesis:
  Yet Another Reo Semantics: Reasoning about Speculative Execution

- Outline:
  1. Background and Challenges
  2. Syntax
     typed language for compositions and components
  3. Semantics
     logical framework with data types and data streams

Overview

- Master's thesis:
  Yet Another Reo Semantics: Reasoning about Speculative Execution

- Outline:
  1. Background and Challenges
  2. Syntax
     typed language for compositions and components
  3. Semantics
     logical framework with data types and data streams
  4. Logical Analysis
     independence, progress, linearity, and causality

Overview

- Master's thesis:
  Yet Another Reo Semantics: Reasoning about Speculative Execution

- Outline:
  1. Background and Challenges
  2. Syntax
     typed language for compositions and components
  3. Semantics
     logical framework with data types and data streams
  4. Logical Analysis
     independence, progress, linearity, and causality

- Throughline:
  Speculative Execution

Overview

- Master's thesis:
  Yet Another Reo Semantics: Reasoning about Speculative Execution

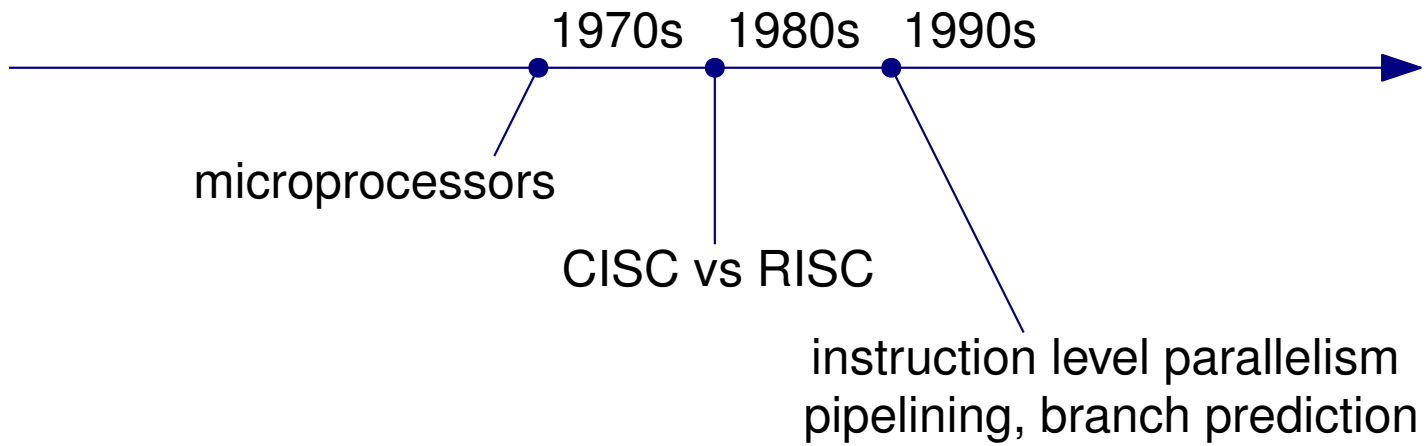
- Outline:
  1. Background and Challenges
  2. Syntax
     typed language for compositions and components
  3. Semantics
     logical framework with data types and data streams
  4. Logical Analysis
     independence, progress, linearity, and causality

- Throughline:
  Speculative Execution

Background and Challenges

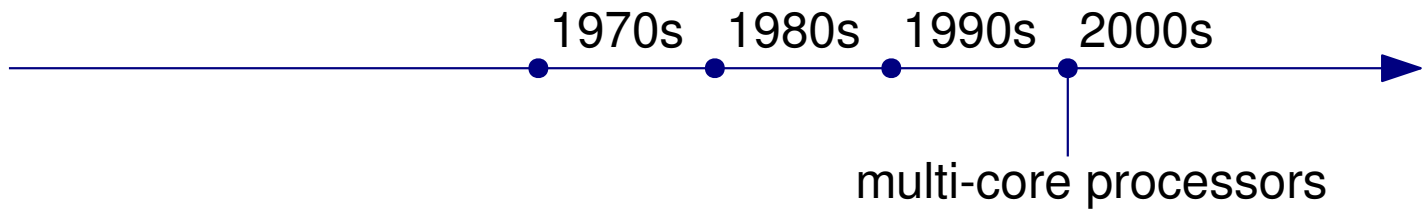
- Increasing demand for computation and communication
- Challenge to meet demand:
  - Increase **throughput**
  - Reduce **energy costs**
- Scalable architecture

Timeline



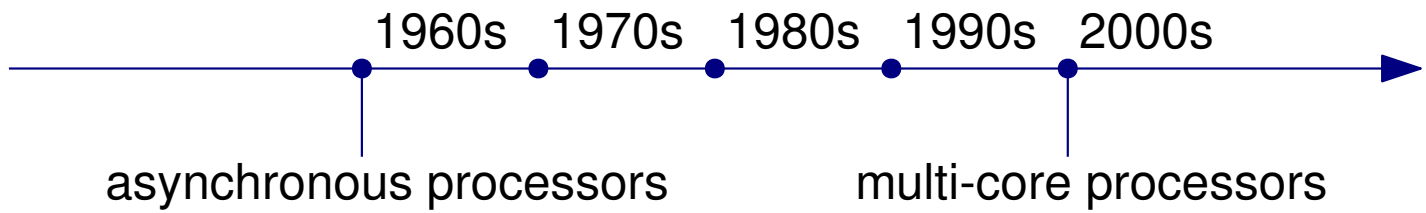
- RISC allows more complex optimizations, e.g. **speculative execution**

Timeline



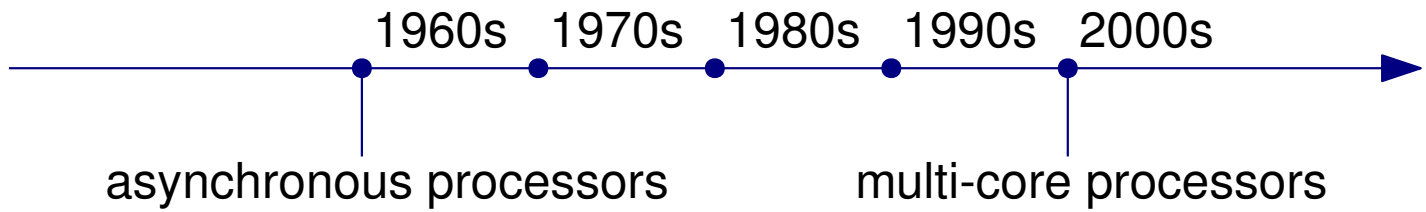
- Multiple processors on single die improve **throughput**
- Bounding clock speeds reduces **energy costs**

Timeline



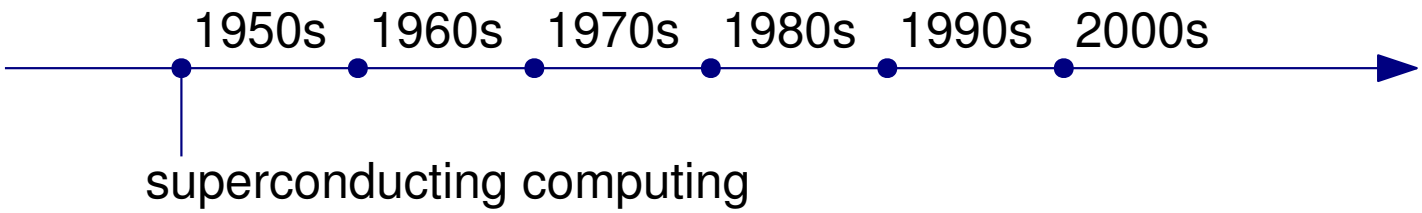
- Challenge of asynchronous processors is **concurrency**
- Concurrency resurrected by multi-core

Timeline



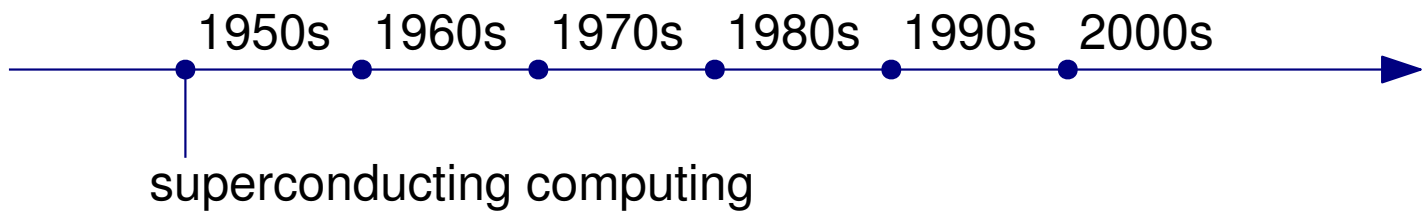
- Delay insensitivity improve **throughput**
- Asynchronous processors reduce **energy costs**

Timeline



- Cool your circuit to 4 degree Kelvin
- Superconductivity: zero electrical resistance on wires
- Josephson junction
- IARPA's Cryogenic Computing Complexity (C3) works towards switching speeds of ~1ns

Timeline

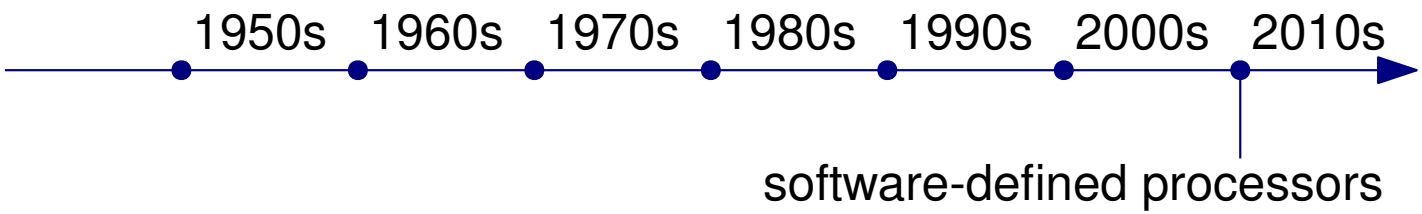


Intuition:

- Electrons flowing by nano pulses
- Computation is controlling the flow
- “The actual usefulness of computation lies in getting rid of the haystack, leaving only the needle.”

cf. Two puzzles about computation, S. Abramsky, 2014
cf. Irreversibility and heat generation in the c. process, R. Landauer, 1961

Timeline



- **compile to hardware**
- Costs of tape-out of integrated circuits has dropped: hundred 1x1mm chips @ 28nm for only $14,000
- Hennessy and Patterson's 2017 Turing Award Lecture

Background and Challenges

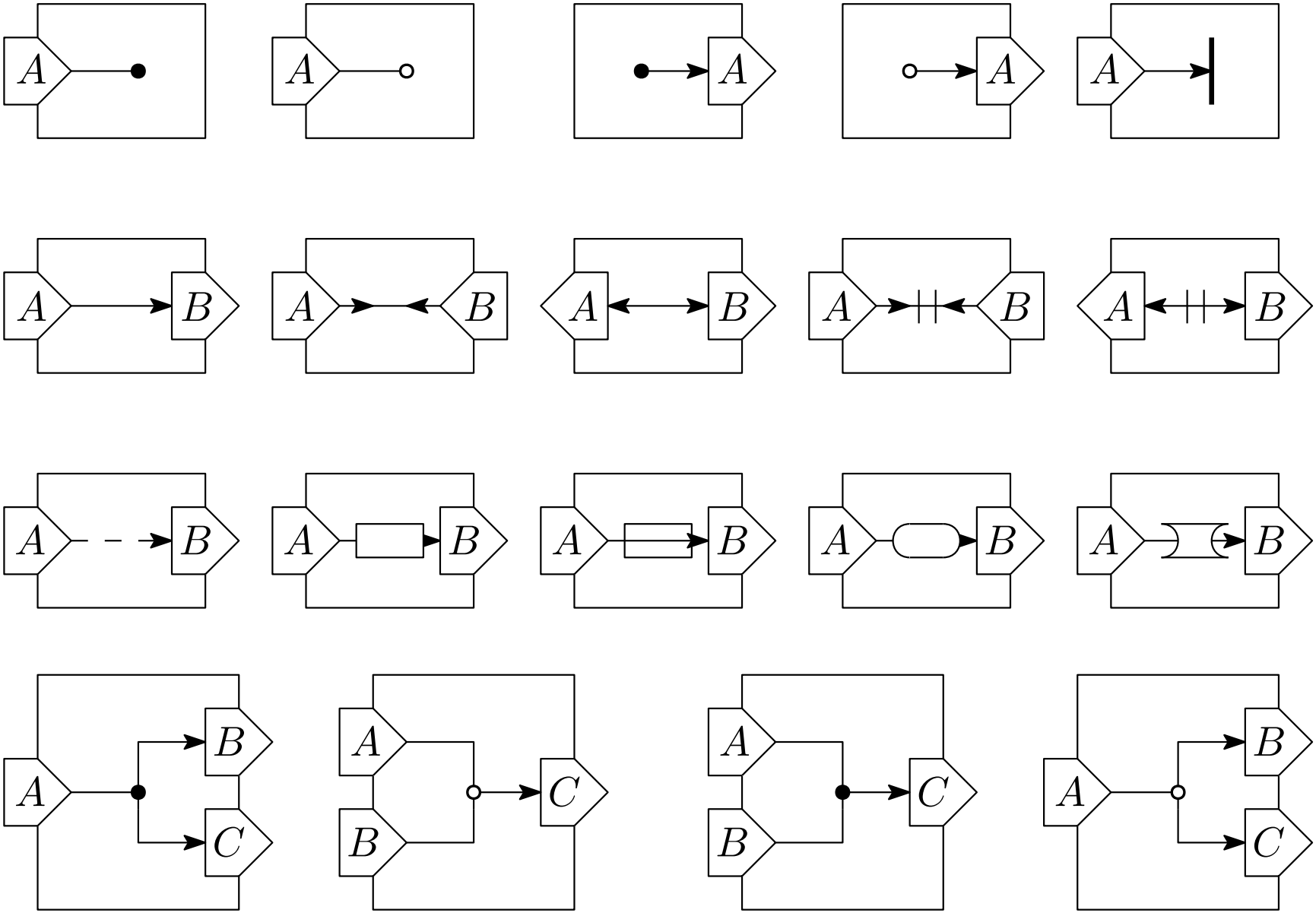
- Scalable architecture
  - 1 CPU $\approx 10^9$ transistors
  - 1 Internet $\approx 10^9$ CPUs
- Modular software correctness
- Polymorphic software scaling
- Reo

Overview

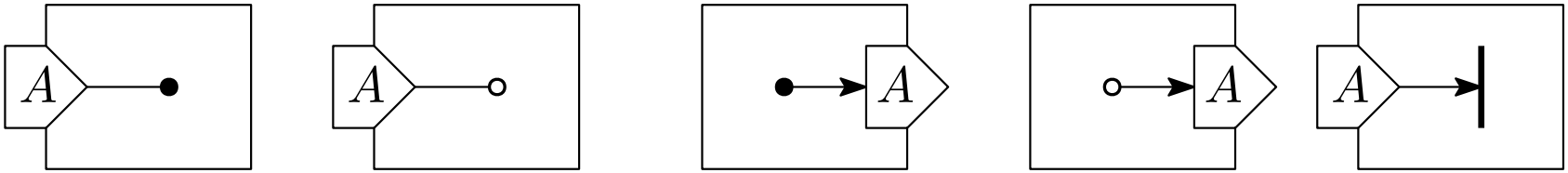
- Master's thesis:
  Yet Another Reo Semantics: Reasoning about Speculative Execution

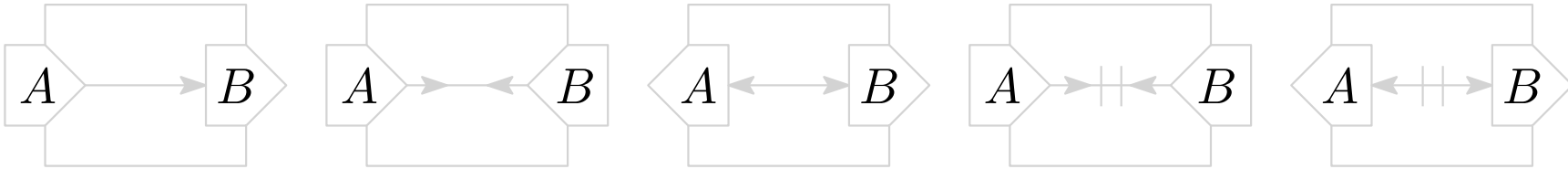
- Outline:
  1. Background and Challenges
  2. *Syntax*
     typed language for compositions and components
  3. Semantics
     logical framework with data types and data streams
  4. Logical Analysis
     independence, progress, linearity, and causality
- Throughline:
  Speculative Execution

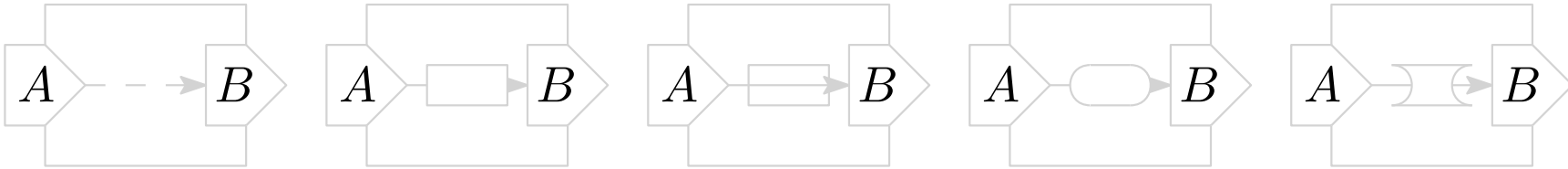
Standard Components

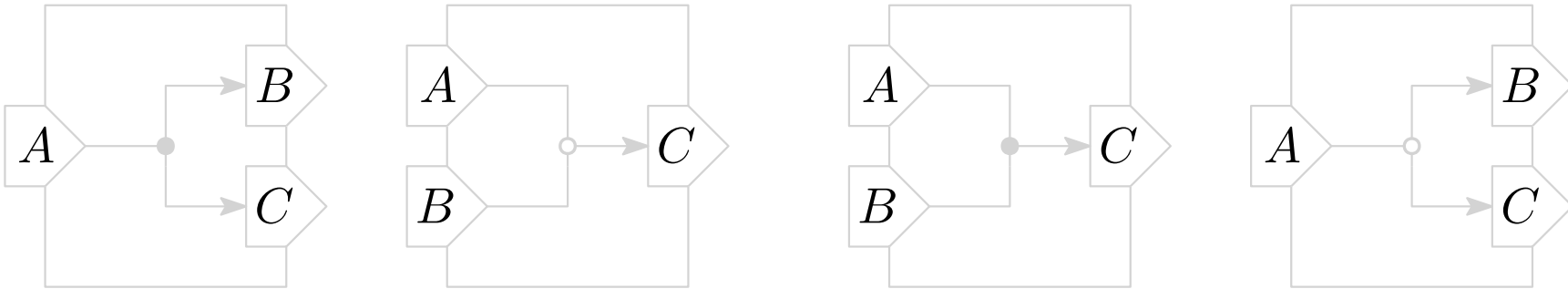


Unary Components

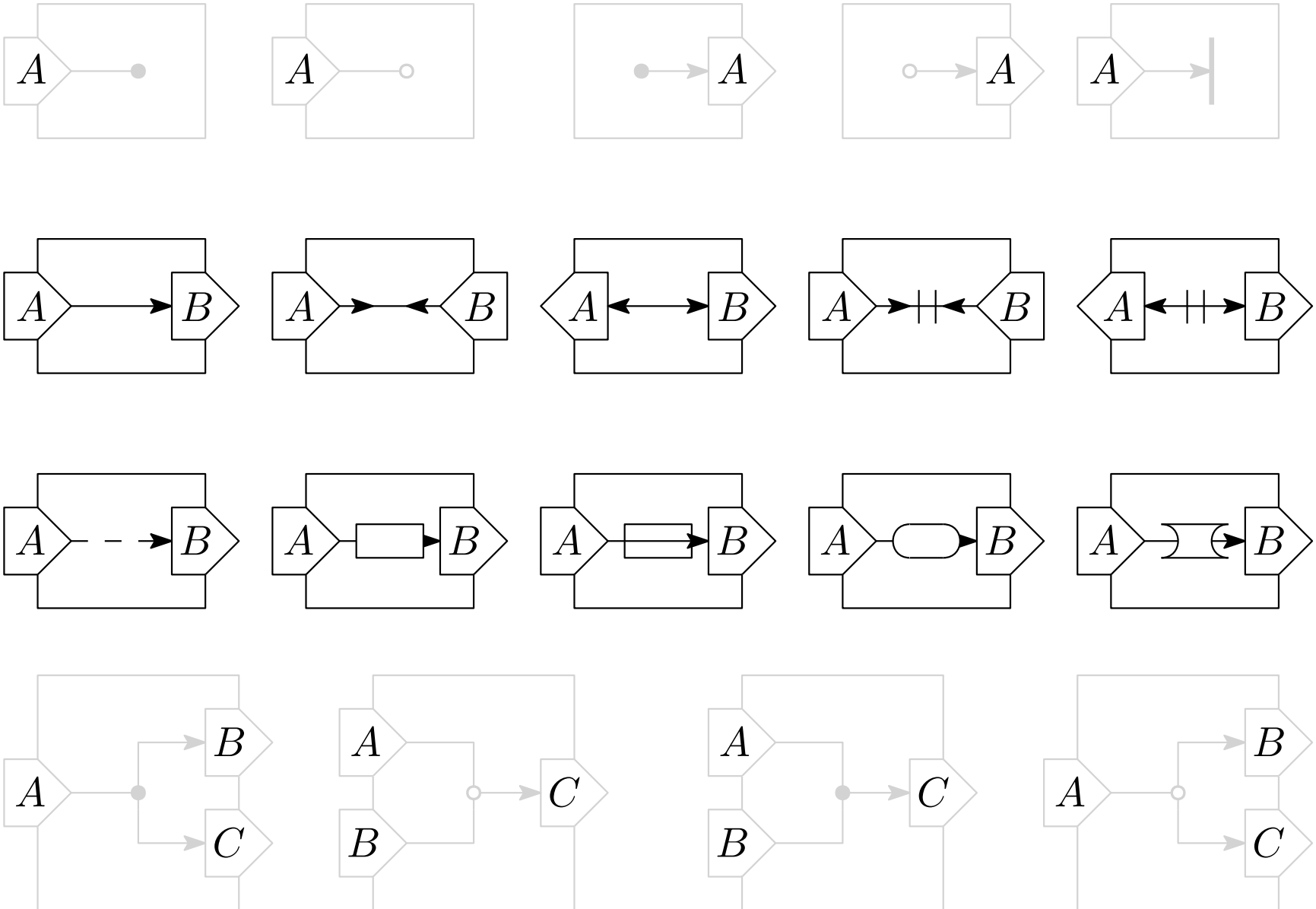




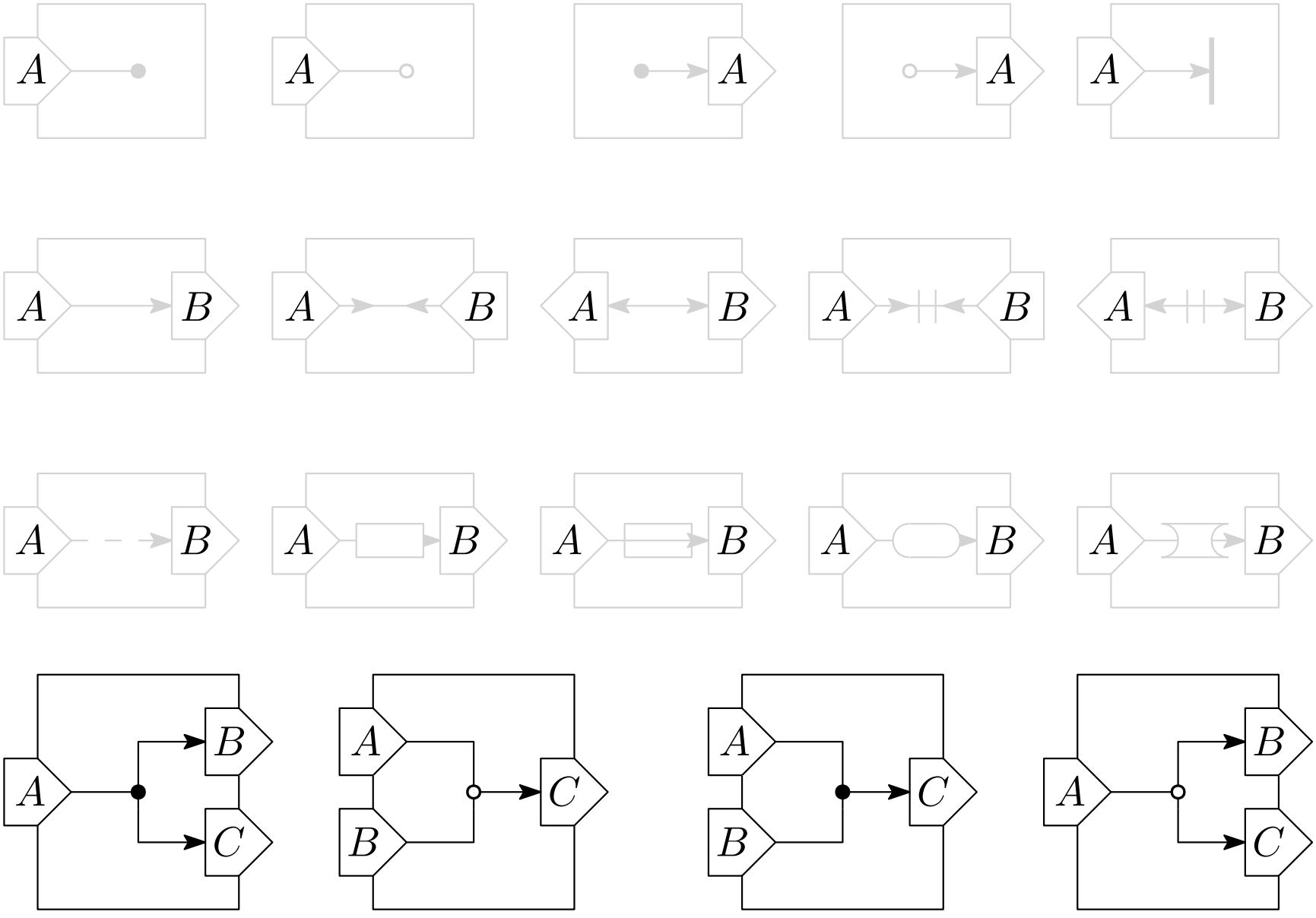




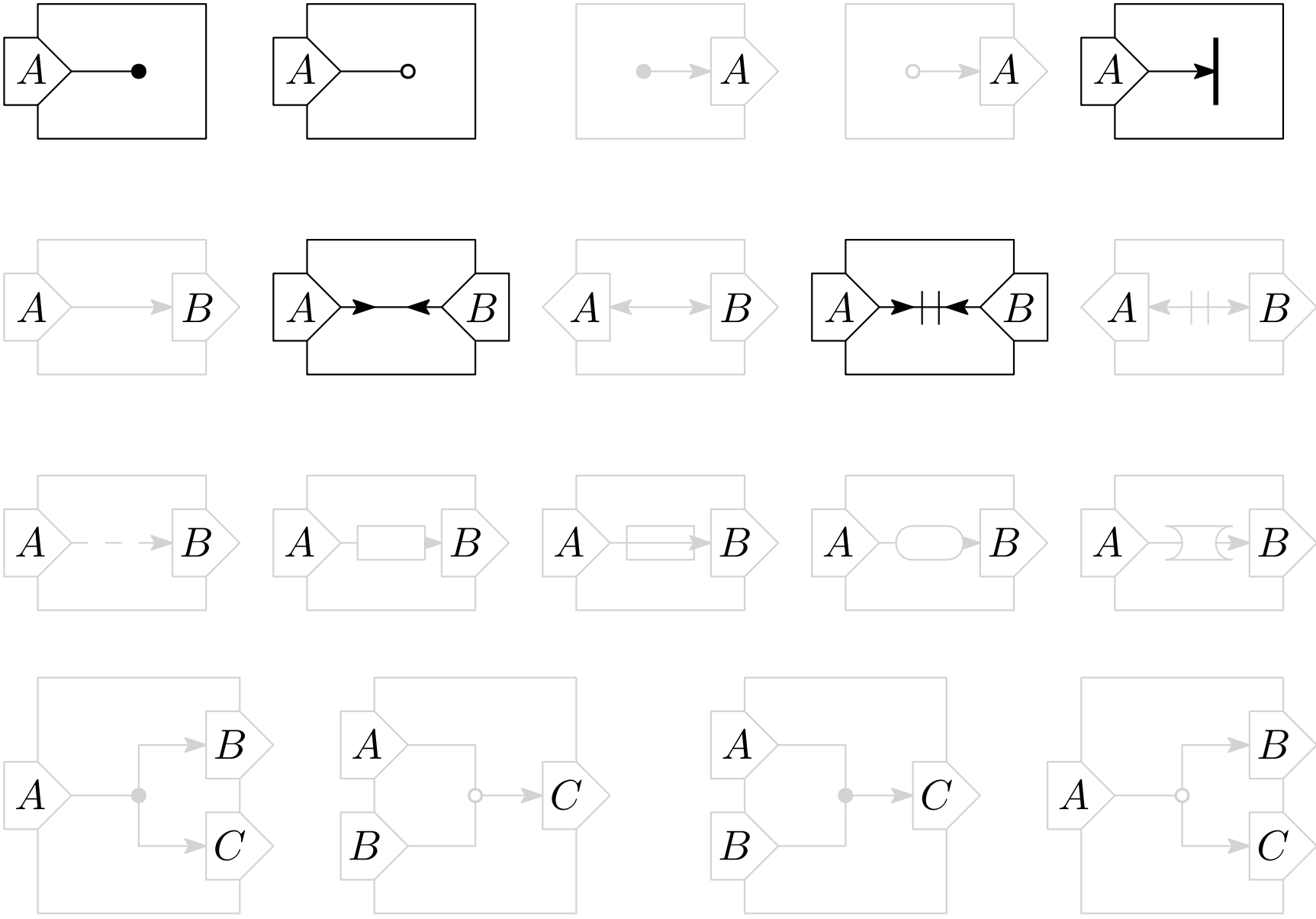
Binary Components



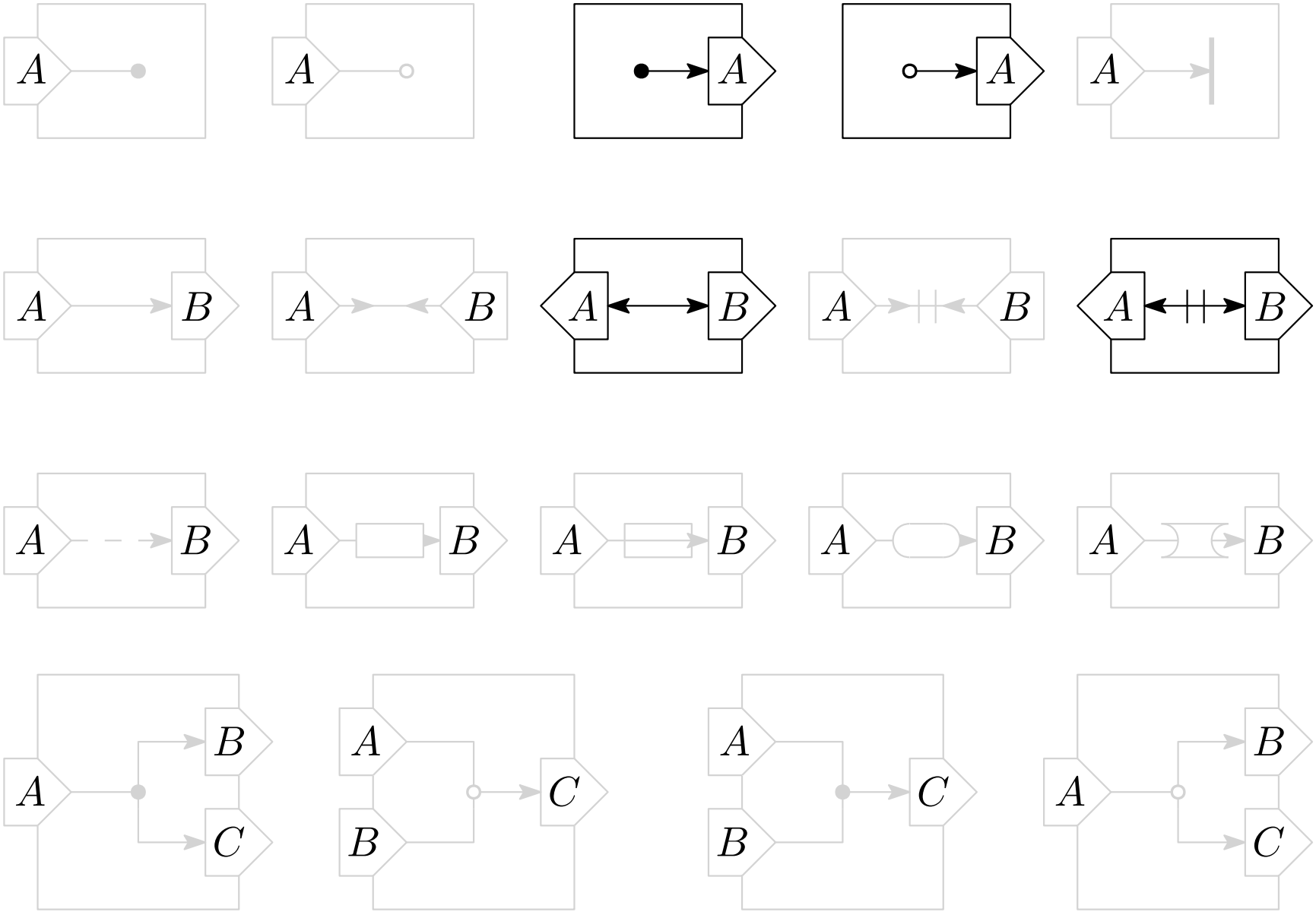
Ternary Components



Input-only Components



Output-only Components

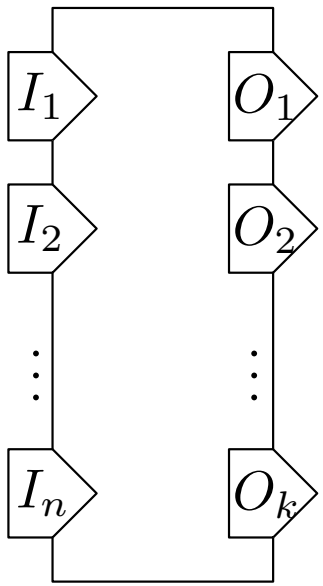


Interface

Definition

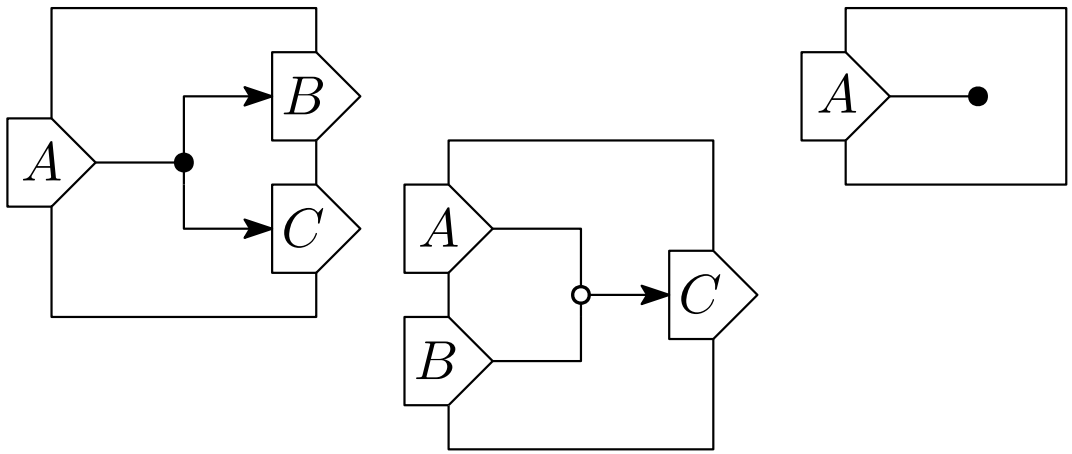
An *interface* $\langle I_1 : \alpha_1, \ldots, I_n : \alpha_n \mid O_1 : \beta_1, \ldots, O_k : \beta_k \rangle$ consists of:

1. A sequence of **input** ports $I_1, \ldots, I_n$
2. A sequence of **ouput** ports $O_1, \ldots, O_k$
3. A type assignment $I_i : \alpha_i$ and $O_j : \beta_j$ for data types α_i, β_j

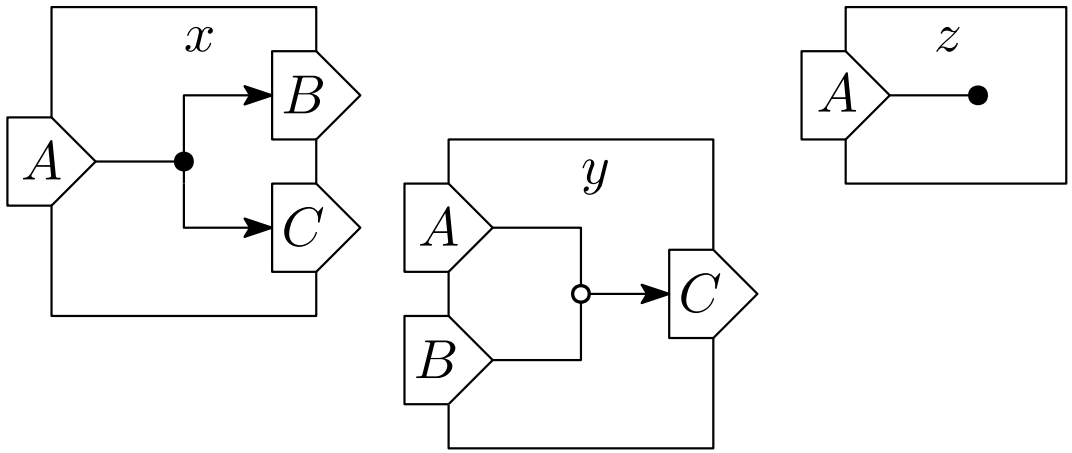


Given interface U, by $U^{\perp}$ we denote its **dual**.

Instances and References

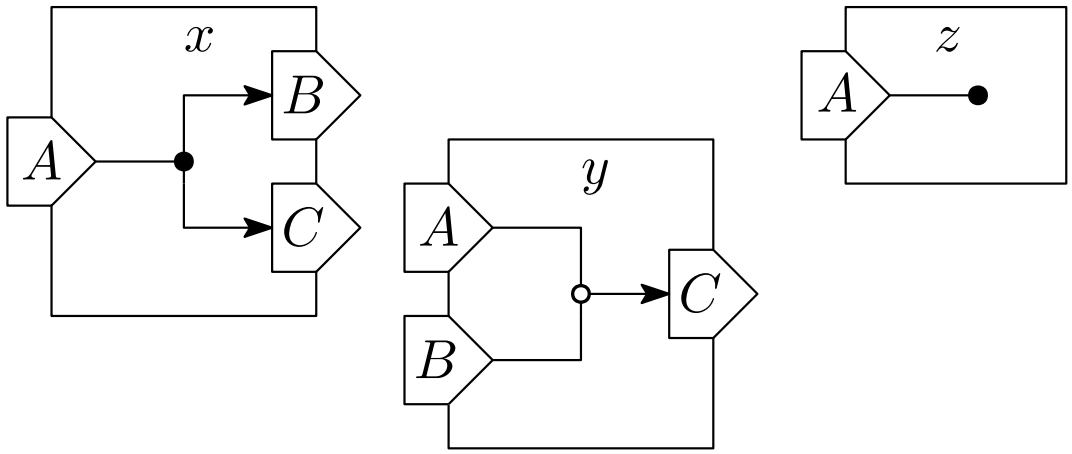


Instances and References



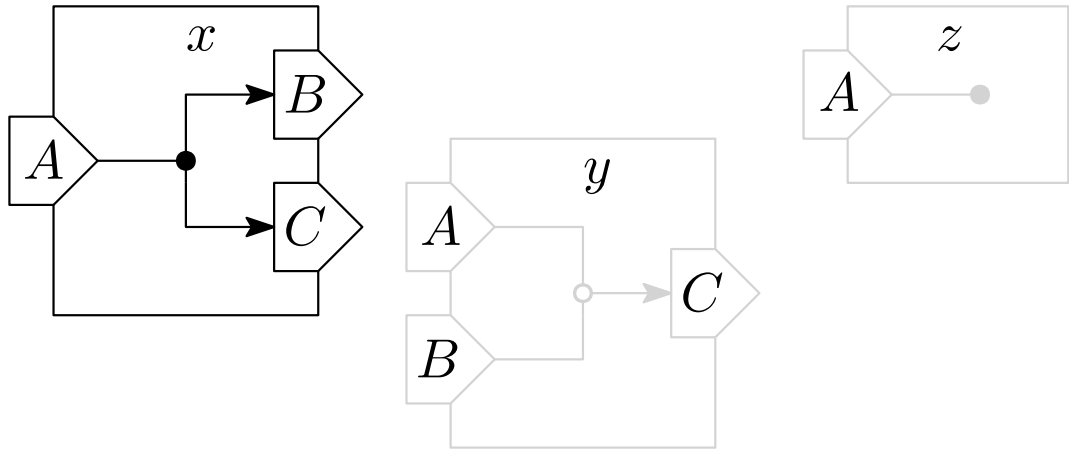
Component instances $x, y, z, \ldots$

Instances and References



Component instances $x, y, z, \ldots$
Qualified $x.A$ and **unqualified** A.

Composition

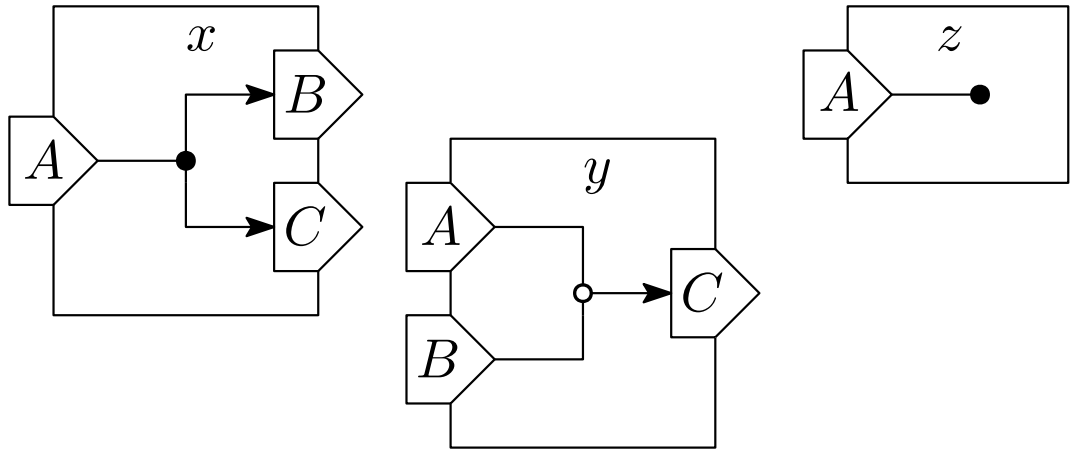


Definition

A *composition* is either:

- an instance

Composition

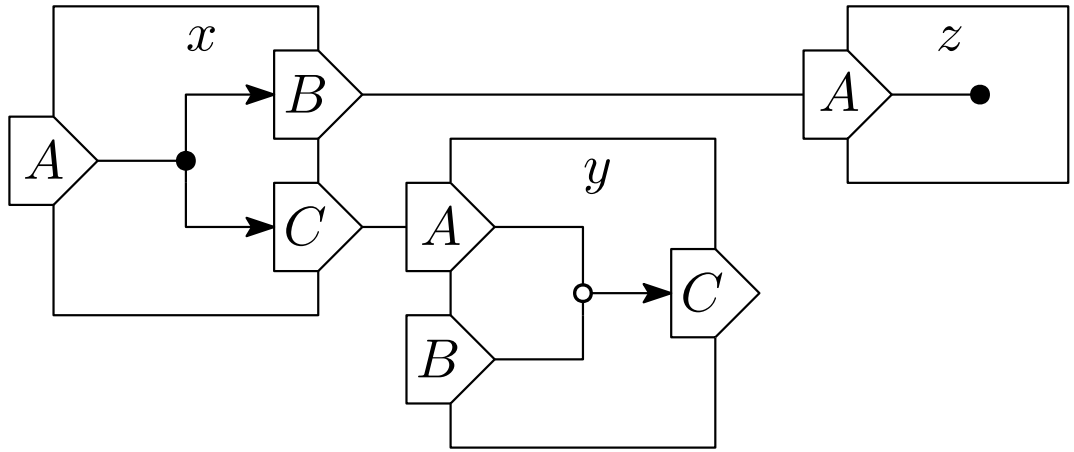


Definition

A *composition* is either:

- an instance
- two compositions adjoined

Composition

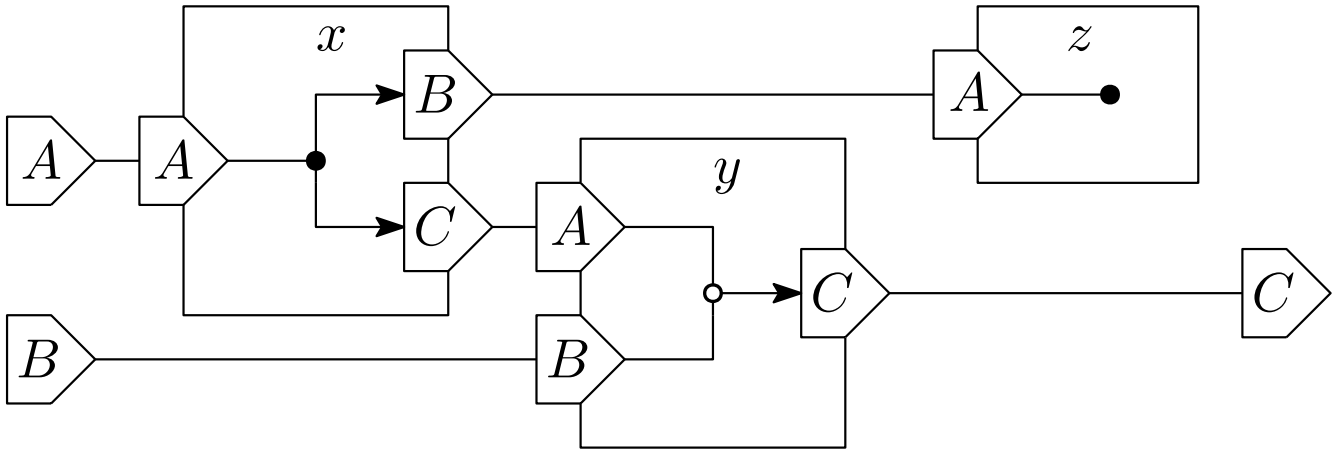


Definition

A *composition* is either:

- an instance
- two compositions adjoined
- identification of two references of a composition

Composition



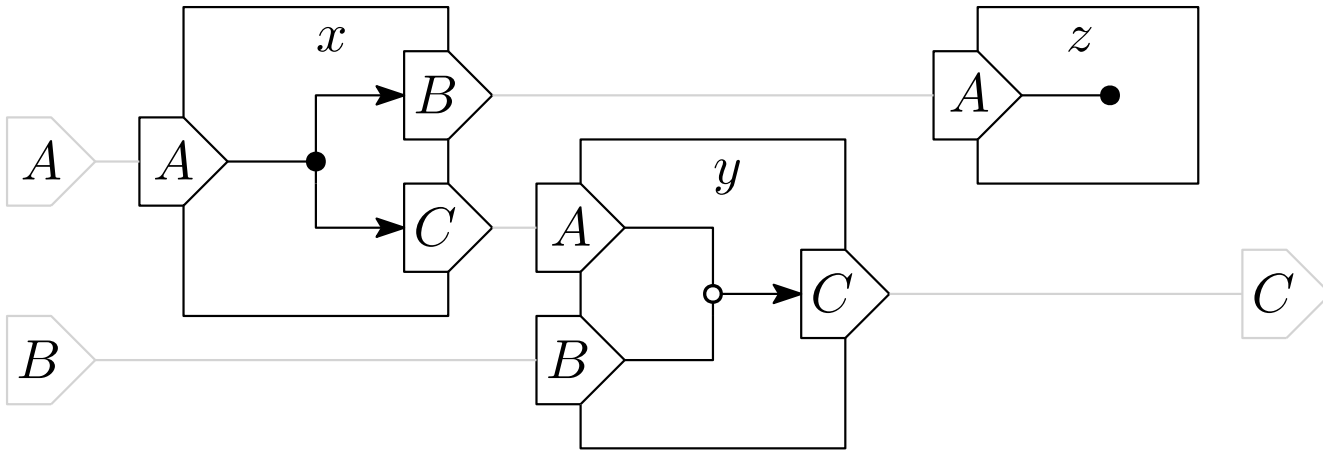
Definition

A *composition* is either:

- an instance
- two compositions adjoined
- identification of two references of a composition

Well-formed if no unqualified references

Component

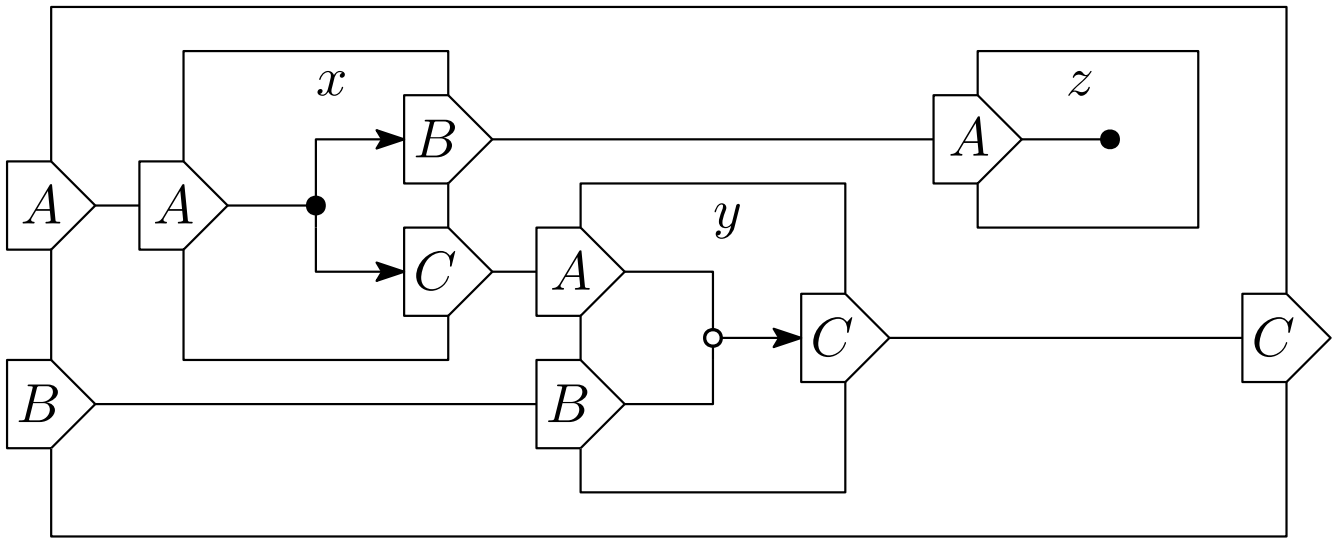


Definition (to be continued)

A *component* is either:

- primitive

Component



Definition (to be continued)

A *component* is either:

- primitive
- composite

Results

1. Formal graphical language for Reo
2. Designed normalization and type checker
3. Implemented prototype (see garbage can)

cf. A survey of graphical languages for monoidal categories,
P. Selinger, 2009

cf. Sequent calculus: a logic and a language for computation and duality,
P. Downen, 2017

Overview

- Master's thesis:
  Yet Another Reo Semantics: Reasoning about Speculative Execution

- Outline:
  1. Background and Challenges
  2. Syntax
     typed language for compositions and components
  3. Semantics
     logical framework with data types and data streams
  4. Logical Analysis
     independence, progress, linearity, and causality
- Throughline:
  Speculative Execution

Data domains

Data types α, β, ...

- contain 'null' value $*$
- are (infinitely) enumerable
- are regularly structured (...)

Examples

$\mathrm{Signal} = \{*, 0\}$ and $\mathrm{Nat} = \{*, 0, 1, 2, \ldots\}$

Stream domains

Given data type α, β, ...
Data streams $(\mathbb{N} \to \alpha)$, $(\mathbb{N} \to \beta)$, ...

- streams are not enumerable

Examples

σ is a data stream over Signal
head $\sigma(0)$ is $*$ or 0
tail σ' is stream derivative

(cf. On Streams and Coinduction, J.J.M.M. Rutten, 2002)

Multi-sorted logic

Sorts:

- each data type is a distinct sort
- each data stream is a distinct sort
- there is a natural sort $\mathbb{N}$

Non-logical symbols:

- $*_\alpha$ null constant
- d_α data constant
- $\bot$ proposition
- $=$ equality predicate
- $0, 1, 2, \ldots$ and $+, -, \times, \leq$ for naturals
- at_α with arity $\langle(\mathbb{N} \to \alpha), \mathbb{N}, \alpha\rangle$

Multi-sorted logic

Terms: standard

- variables x^s
- constants
- function symbols

Formulas: standard

- propositions
- predicates of terms
- standard logical symbols $\neg$, $\wedge$, $\vee$, $\rightarrow$
- first-order quantification $\exists x^s$, $\forall x^s$ binders for x^s

We write $\mathrm{at}(X, t)$ as $X(t)$, called *applications*.

Definition

A *coordination protocol* is a first-order formula such that all free variables X^s are of a data stream sort $s = (\mathbb{N} \rightarrow \alpha)$

Coordination protocols

Coordination protocols induce a set of infinite tables
(cf. Rule-Based Form for Stream Constraints, K. Dokter, 2018)

Examples

Let X be a port of data type Signal.
Consider $\phi = \forall t.(X(t) = * \vee X(t) = 0)$
Solution: assignment of X to a data stream that sats. formula

$$\mathscr{L}(\phi) = \left\{ \begin{array}{cc} t & X \\ \hline 0 & * \\ 1 & * \\ 2 & * \\ \vdots & \vdots \end{array} \right., \ldots, \left\{ \begin{array}{cc} t & X \\ \hline 0 & * \\ 1 & 0 \\ 2 & * \\ \vdots & \vdots \end{array} \right., \ldots, \left\{ \begin{array}{cc} t & X \\ \hline 0 & 0 \\ 1 & 0 \\ 2 & 0 \\ \vdots & \vdots \end{array} \right., \ldots$$

Coordination protocols

Fact: coordination protocols only 'interact' for shared variables

Examples (No interaction)

Consider $\phi = \forall t.(X(t) = *)$ and $\psi = \forall t.(Y(t) = 0)$

$$\mathscr{L}(\phi) = \left\{ \begin{array}{|cc|} \hline t & X \\ \hline 0 & * \\ 1 & * \\ \vdots & \vdots \\ \hline \end{array} \right\} \text{ and } \mathscr{L}(\psi) = \left\{ \begin{array}{|cc|} \hline t & Y \\ \hline 0 & 0 \\ 1 & 0 \\ \vdots & \vdots \\ \hline \end{array} \right\}$$

Consider

$$\mathscr{L}(\phi \wedge \psi) = \left\{ \begin{array}{|ccc|} \hline t & X & Y \\ \hline 0 & * & 0 \\ 1 & * & 0 \\ \vdots & \vdots & \vdots \\ \hline \end{array} \right\}$$

Coordination protocols

Fact: coordination protocols only 'interact' for shared variables

Examples (Interaction)

Consider $\phi = \forall t.(X(t) = *) \vee \forall t.(X(t) = 0)$ and $\psi = \exists t.(X(t) = 0)$

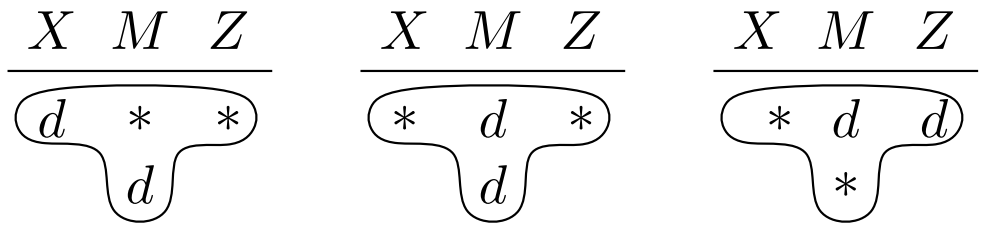
$$\mathscr{L}(\phi) = \left\{ \begin{array}{cc} t & X \\ \hline 0 & * \\ 1 & * \\ \vdots & \vdots \end{array}, \begin{array}{cc} t & X \\ \hline 0 & 0 \\ 1 & 0 \\ \vdots & \vdots \end{array} \right\} \text{ and } \mathscr{L}(\psi) = \left\{ \begin{array}{cc} t & X \\ \hline 0 & * \\ 1 & 0 \\ \vdots & \vdots \end{array}, \begin{array}{cc} t & X \\ \hline 0 & 0 \\ 1 & * \\ \vdots & \vdots \end{array} \right\}$$

Consider

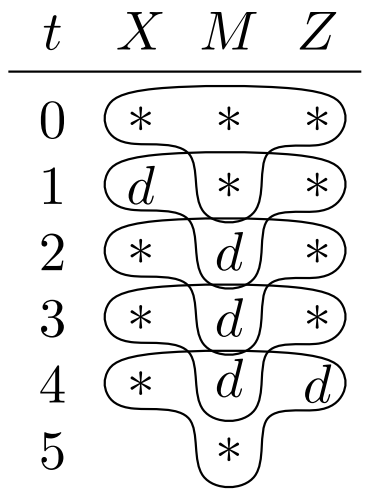
$$\mathscr{L}(\phi \wedge \psi) = \left\{ \begin{array}{cc} t & X \\ \hline 0 & 0 \\ 1 & 0 \\ \vdots & \vdots \end{array} \right\}$$

Frame conditions

Insight: modeling of buffers using frame conditions.

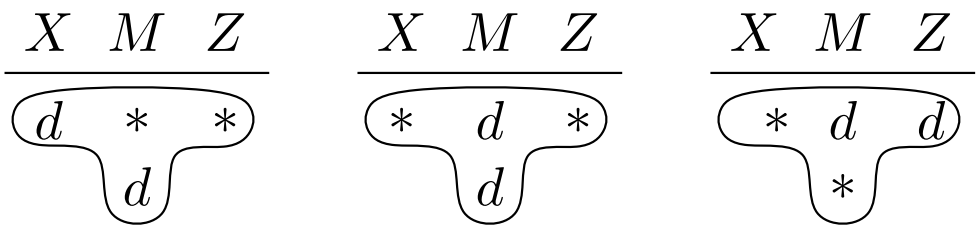


These are overlapping as follows:



Frame conditions

Result: definition of buffer

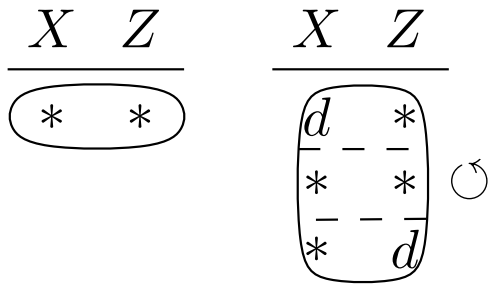


Examples

$$\begin{aligned}\forall t.((\quad & Z(t) = * \quad \wedge M(t) = * \wedge M(t+1) = X(t)) \ \vee \\ (X(t) = * \wedge & Z(t) = * \quad \wedge M(t) \neq * \wedge M(t+1) = M(t)) \vee \\ (X(t) = * \wedge & Z(t) = M(t) \wedge M(t) \neq * \wedge M(t+1) = *))\end{aligned}$$

Frame conditions

Problem: column M is not a port. Can we get rid of it?



Examples

$$\forall t.(\ Z(t) = * \wedge X(t) = * \vee$$
$$(Z(t) = * \wedge \exists j.t < j \wedge X(j) = * \wedge Z(j) = X(t) \wedge$$
$$\forall i.t < i \wedge i < j \rightarrow X(i) = * \wedge Z(i) = *) \vee$$
$$(X(t) = * \wedge \exists j.j < t \wedge X(j) = Z(t) \wedge Z(j) = * \wedge$$
$$\forall i.j < i \wedge i < t \rightarrow X(i) = * \wedge Z(i) = *))$$

Components revisited

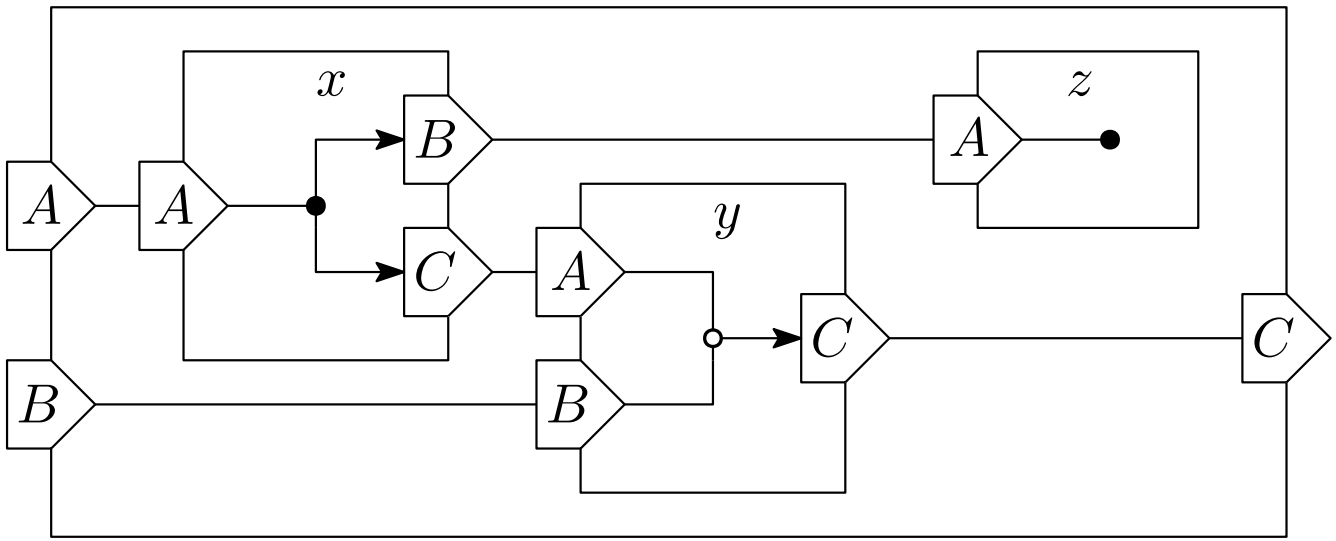
Definition

A *component* $U\phi$ is an interface U and coordination protocol ϕ.

We lift composite constructions to this definition.
Primitive components given as $U\phi$.
Composite components by induction:

- Adjoined components $U\phi$ and $V\psi$:
  take $\phi \wedge \psi$
- Identification of references X and Y on $U\phi$:
  take $\phi \wedge \forall t.(X(t) = Y(t))$

Components revisited



$x :: \langle A : \alpha \mid B : \alpha, C : \alpha \rangle$

$\forall t.(A(t) = B(t) \wedge B(t) = C(t))$

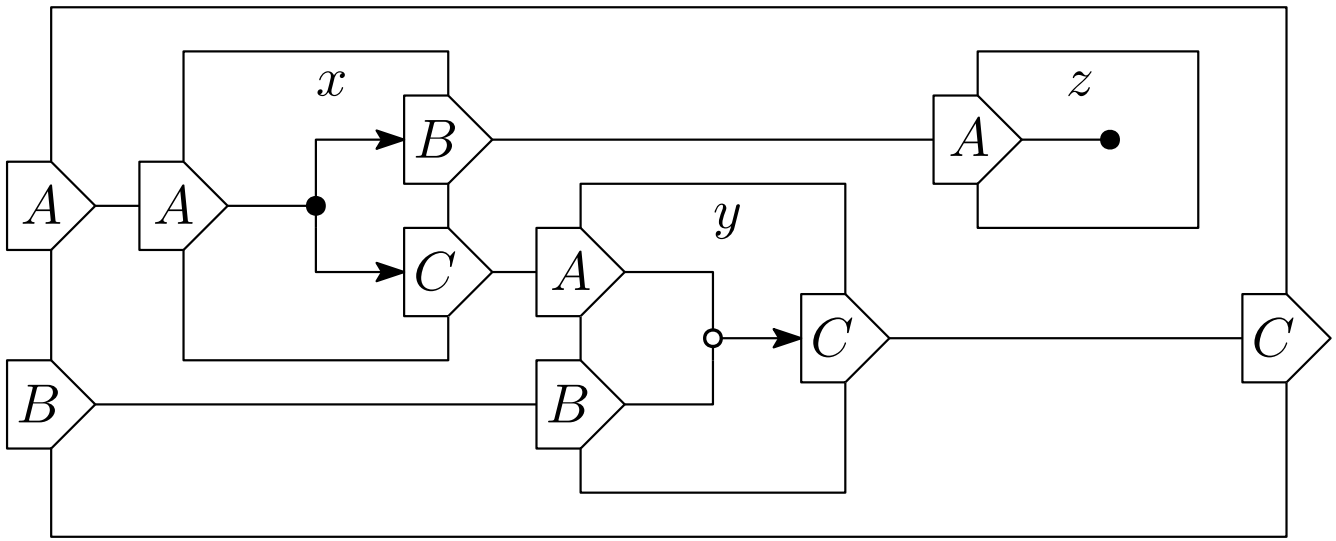
$y :: \langle A : \alpha, B : \alpha \mid C : \alpha \rangle$

$\forall t.((A(t) = C(t) \wedge B(t) = *) \vee (B(t) = C(t) \wedge A(t) = *))$

$z :: \langle A : \alpha \mid \rangle$

$\forall t.(A(t) = * \vee A(t) \neq *)$

Components revisited



$\langle A : \alpha, B : \alpha \mid C : \alpha \rangle$

$\forall t.(x.A(t) = x.B(t) \wedge x.B(t) = x.C(t)) \wedge$

$\forall t.((y.A(t) = y.C(t) \wedge y.B(t) = *) \vee$

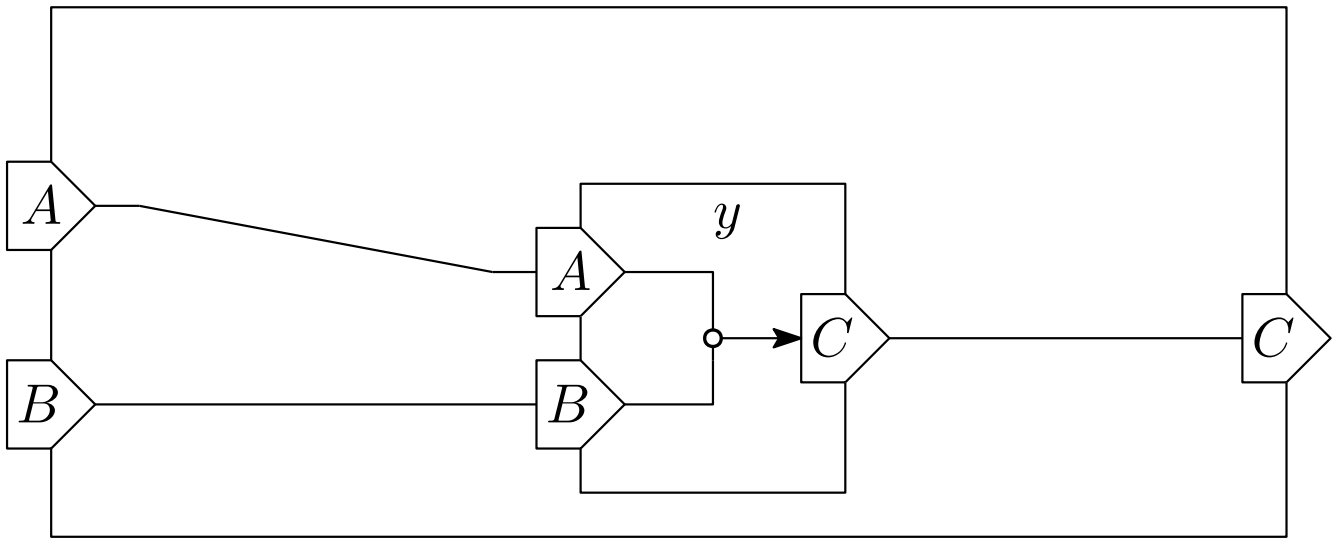
$\quad (y.B(t) = y.C(t) \wedge y.A(t) = *)) \wedge$

$\forall t.(z.A(t) = * \vee z.A(t) \neq *) \wedge$

$\forall t.(x.B(t) = z.A(t)) \wedge \forall t.(x.C(t) = y.A(t)) \wedge$

$\forall t.(A(t) = x.A(t)) \wedge \forall t.(B(t) = y.B(t)) \wedge \forall t.(y.C(t) = C(t))$

Components revisited

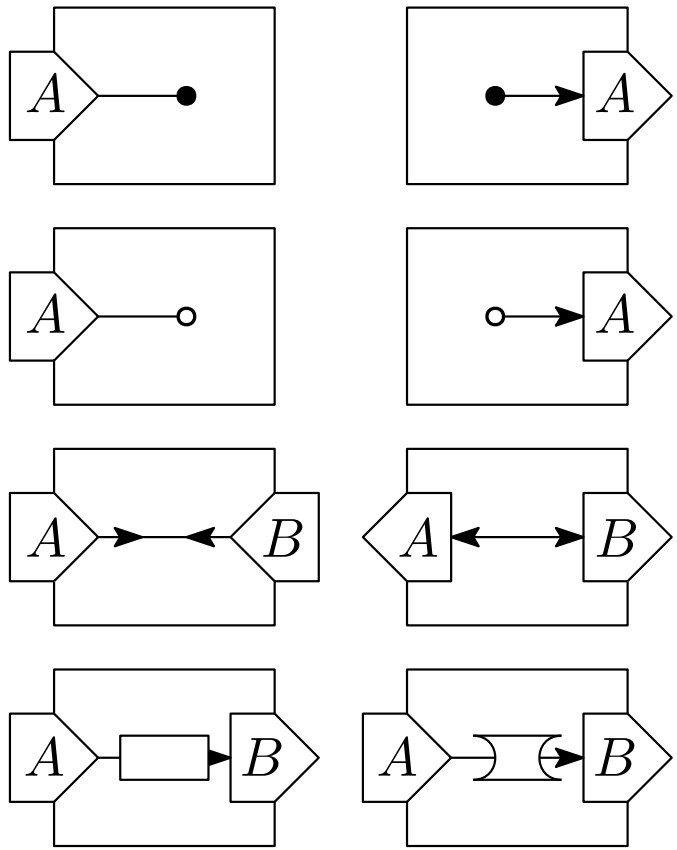


$$\langle A : \alpha, B : \alpha \mid C : \alpha \rangle$$
$$\forall t.((A(t) = C(t) \wedge B(t) = *) \vee$$
$$(B(t) = C(t) \wedge A(t) = *))$$

Duals

Definition

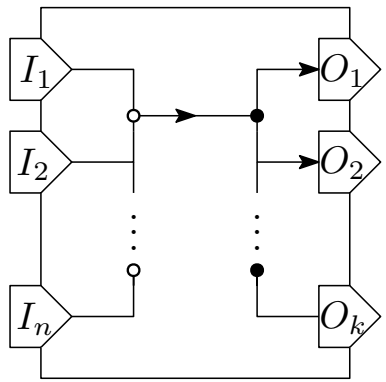
A *dual* of a component $U\phi$ is $U^{\perp}\phi$.



Results

1. Found a suitable logical framework
   with data types and data streams
   and defined primitives

2. Two different kinds of nodes (Reo has the one below)



1. Prophet is dual of buffer

cf. Rule-based form for stream constraints, K. Dokter, 2018
cf. The existence of refinement mappings, M. Abadi & L. Lamport, 1988

Overview

- Master's thesis:
  Yet Another Reo Semantics: Reasoning about Speculative Execution

- Outline:
  1. Background and Challenges
  2. Syntax
     typed language for compositions and components
  3. Semantics
     logical framework with data types and data streams
  4. Logical Analysis
     independence, progress, linearity, and causality
- Throughline:
  Speculative Execution

Speculative Execution

A *speculation* predicts a future state.

- A *true speculation* if future is predicted.
- A *false speculation* if not.

Two methods:

- Parallel isolated worlds, discarding false worlds
- Branch prediction + backtracking

Interesting properties

- Independence
  (delay insensitivity)
- Linearity
  (superconducting/reversibility)
- Causality
  (speculative execution)
- Instantaneous
  (stateless)
- Synchronous and Asynchronous
  (clocked and unclocked)
- Termination and Progress
  (safety and liveness)

Independence

Semantic property

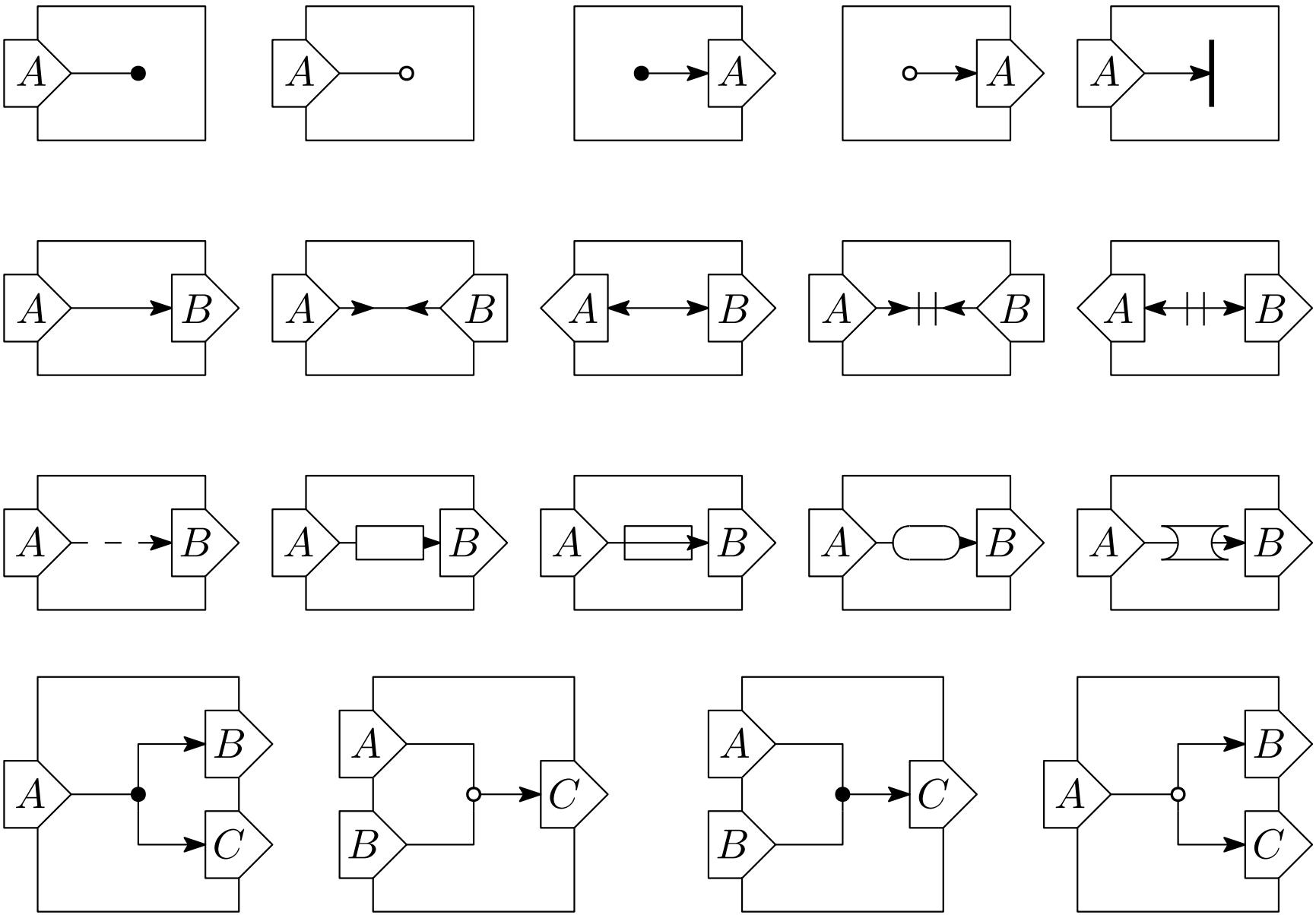
Independence is a closure condition on semantics.

$$\mathscr{L}(\phi) = \left\{ \begin{array}{cc} t & X \\ \hline 0 & * \\ 1 & * \\ \vdots & \vdots \end{array}, \cdots, \begin{array}{cc} t & X \\ \hline 0 & 0 \\ 1 & * \\ \vdots & \vdots \end{array}, \cdots \right.$$

- May remove rows with all $*$
- May insert row with all $*$ before row without all $*$

1. No real-time clocks (row depending on t).
2. Independence is compositional

Independence



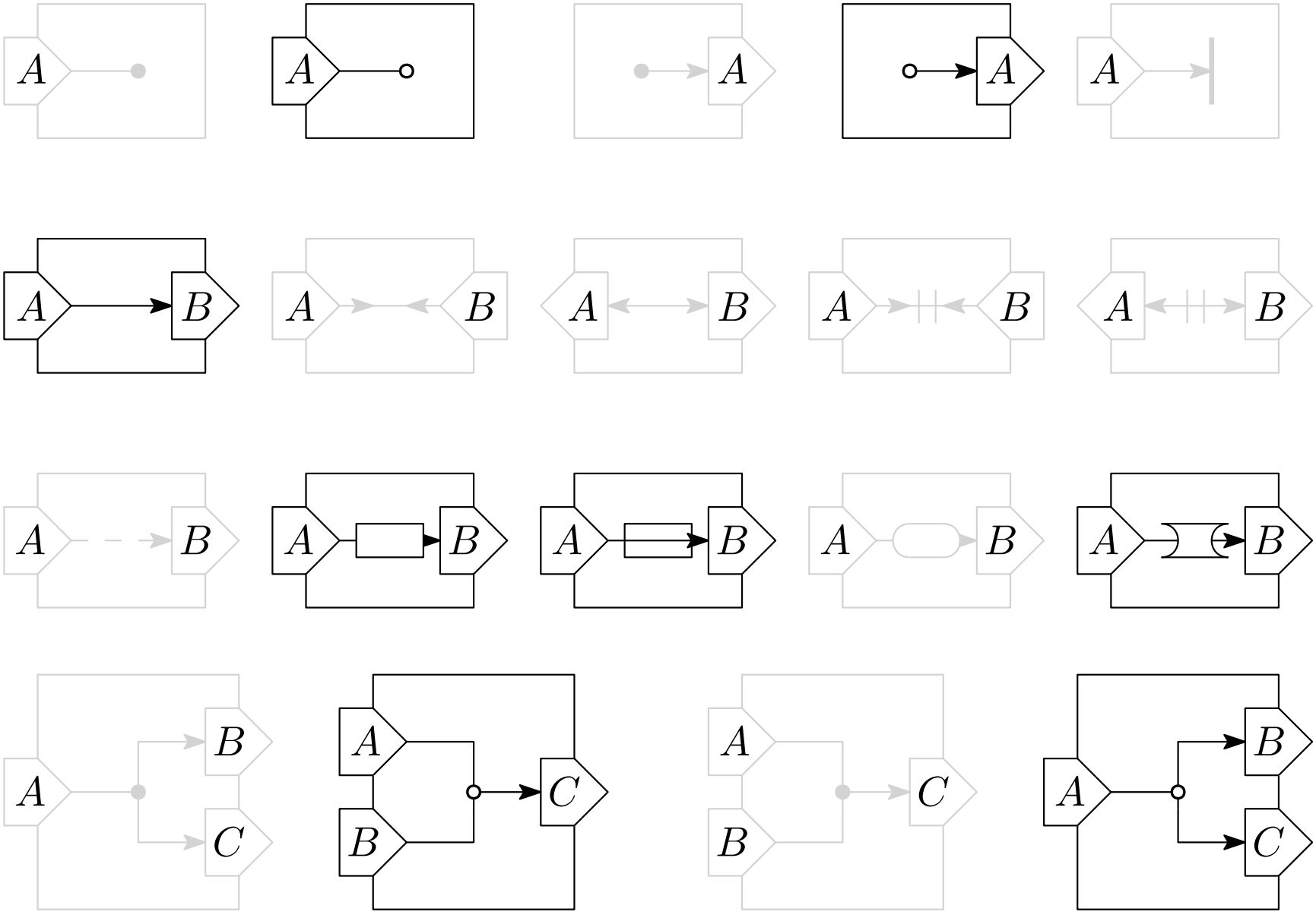
Linearity

Semantic property

Bijection between *input events*, *output events*

t	X	Y
0	$*$	d
1	d	$*$
2	e	e
3	f	$*$
4	$*$	f
$\vdots$	$\vdots$	

Linearity



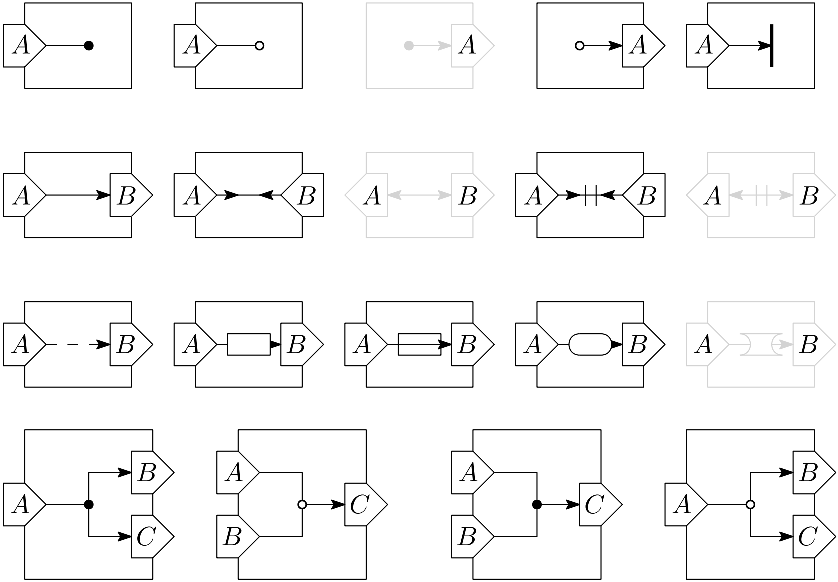
Causality

Semantic property

Every output event at time t related with input event at time $\leq t$.

t	X	Y
0	$*$	$*$
1	d	$*$
2	$*$	d
3	f	d
4	$*$	f
$\vdots$	$\vdots$	

Causality



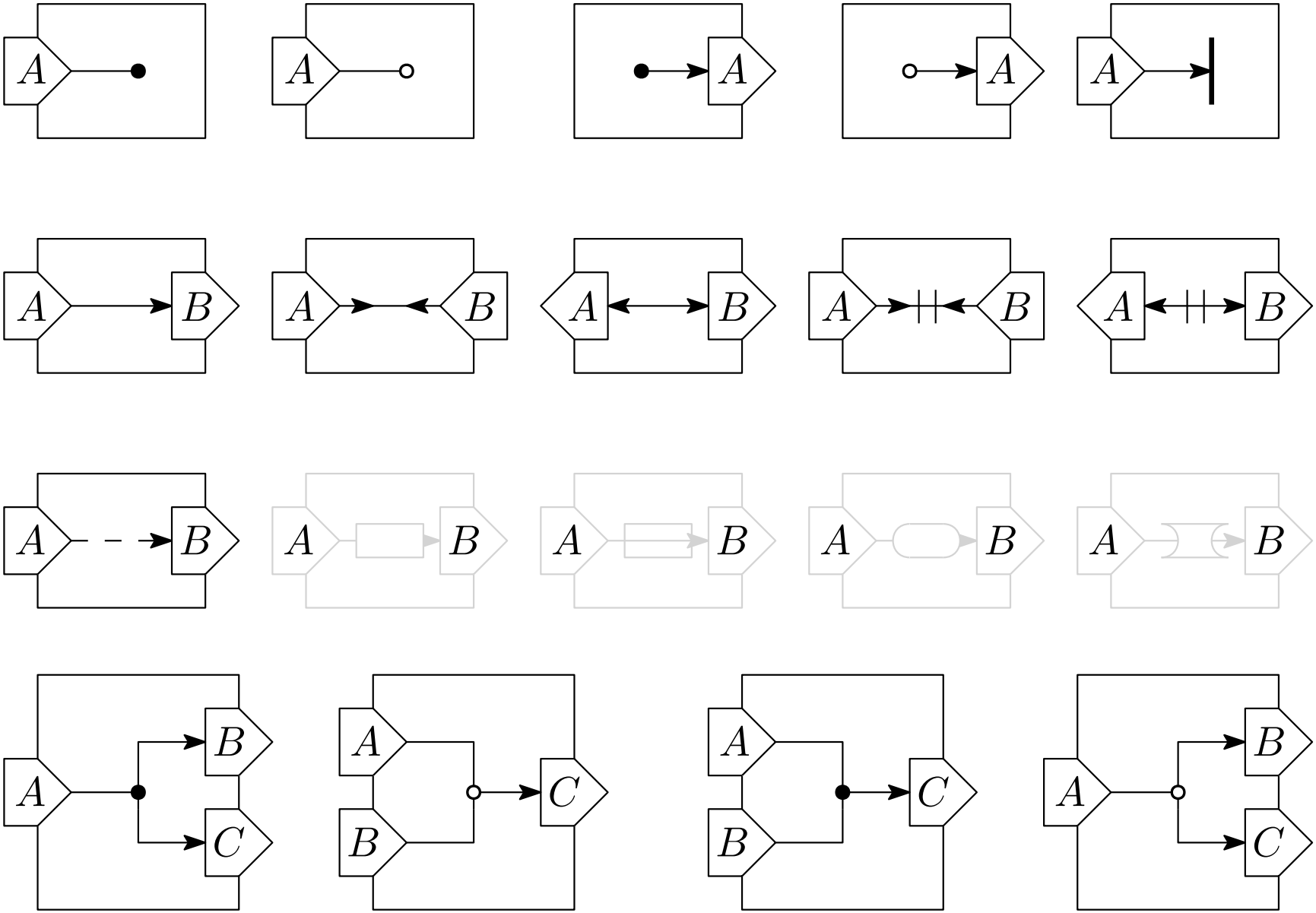
Instantaneous

Syntactic property
Every formula of the shape:

$$\forall t.\phi$$

where ϕ only consists of applications with t.

Instantaneous



Synchronous

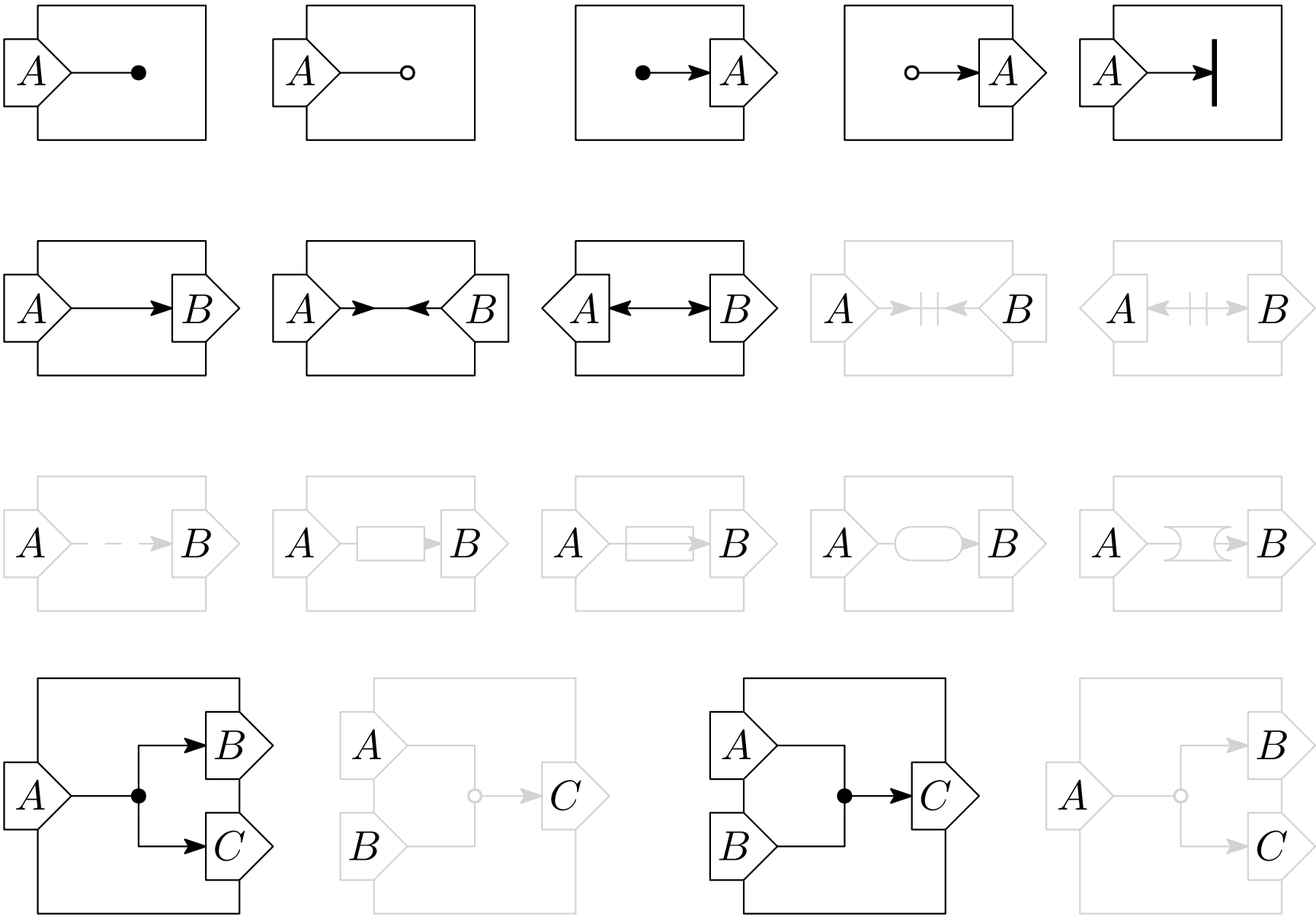
Logical property

A set of port is synchronous if all fire together, or none fire.

Let $P = \{X_1, \ldots, X_n\}$

$$\forall t.((X_1(t) = * \wedge \cdots \wedge X_n(t) = *) \vee (X_1(t) \neq * \wedge \cdots \wedge X_n(t) \neq *))$$

Synchronous



Asynchronous

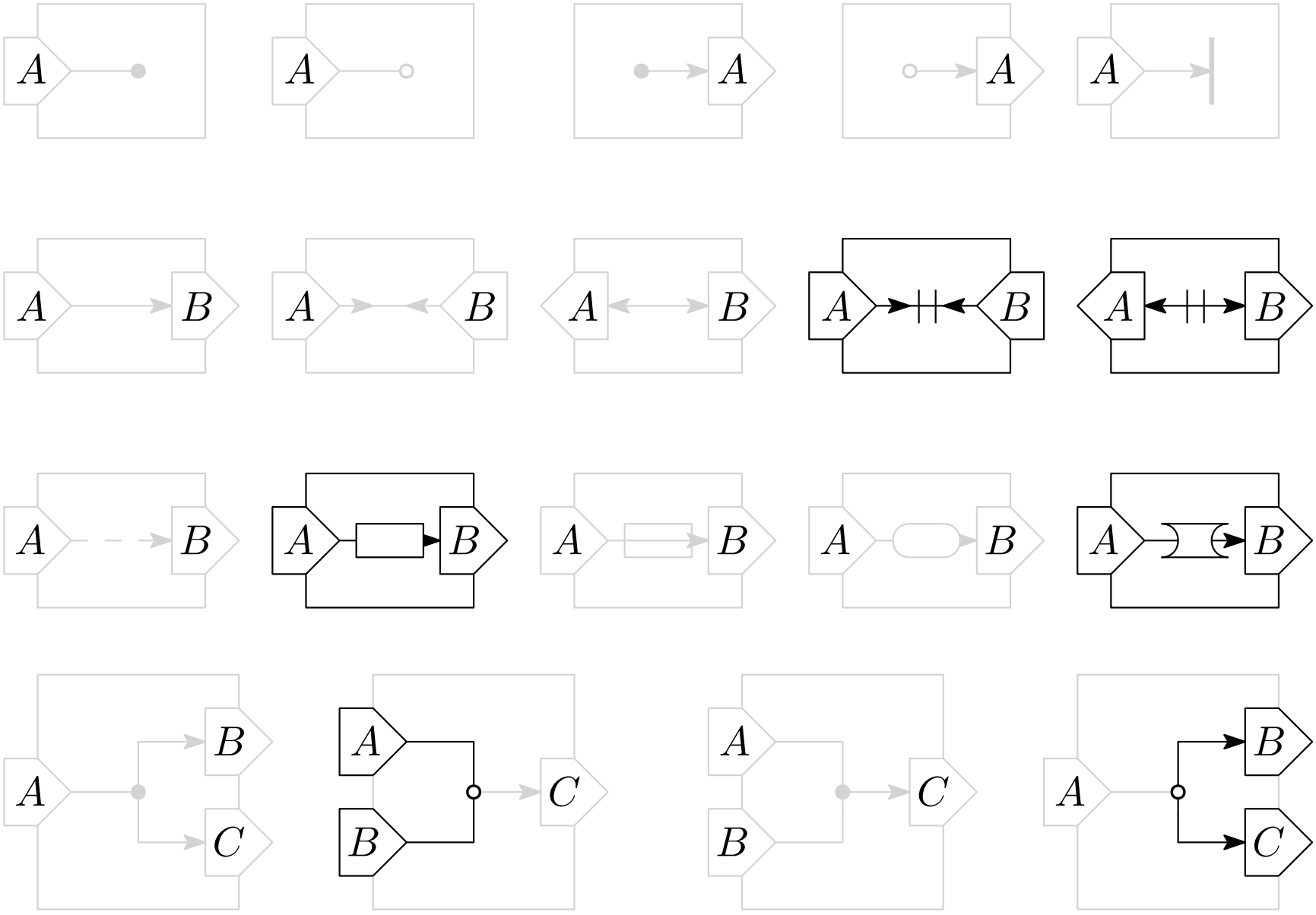
Logical property

A set of ports is asynchronous if at most one port fires.

Let $P = \{X_1, \ldots, X_n\}$

$$\forall t. \bigwedge_{i \in \{1,\ldots,n\}} \left(X_i(t) \neq * \rightarrow \bigwedge_{j \in \{1,\ldots,n\} \setminus \{i\}} X_j(t) = * \right)$$

Asynchronous



Termination and Progress

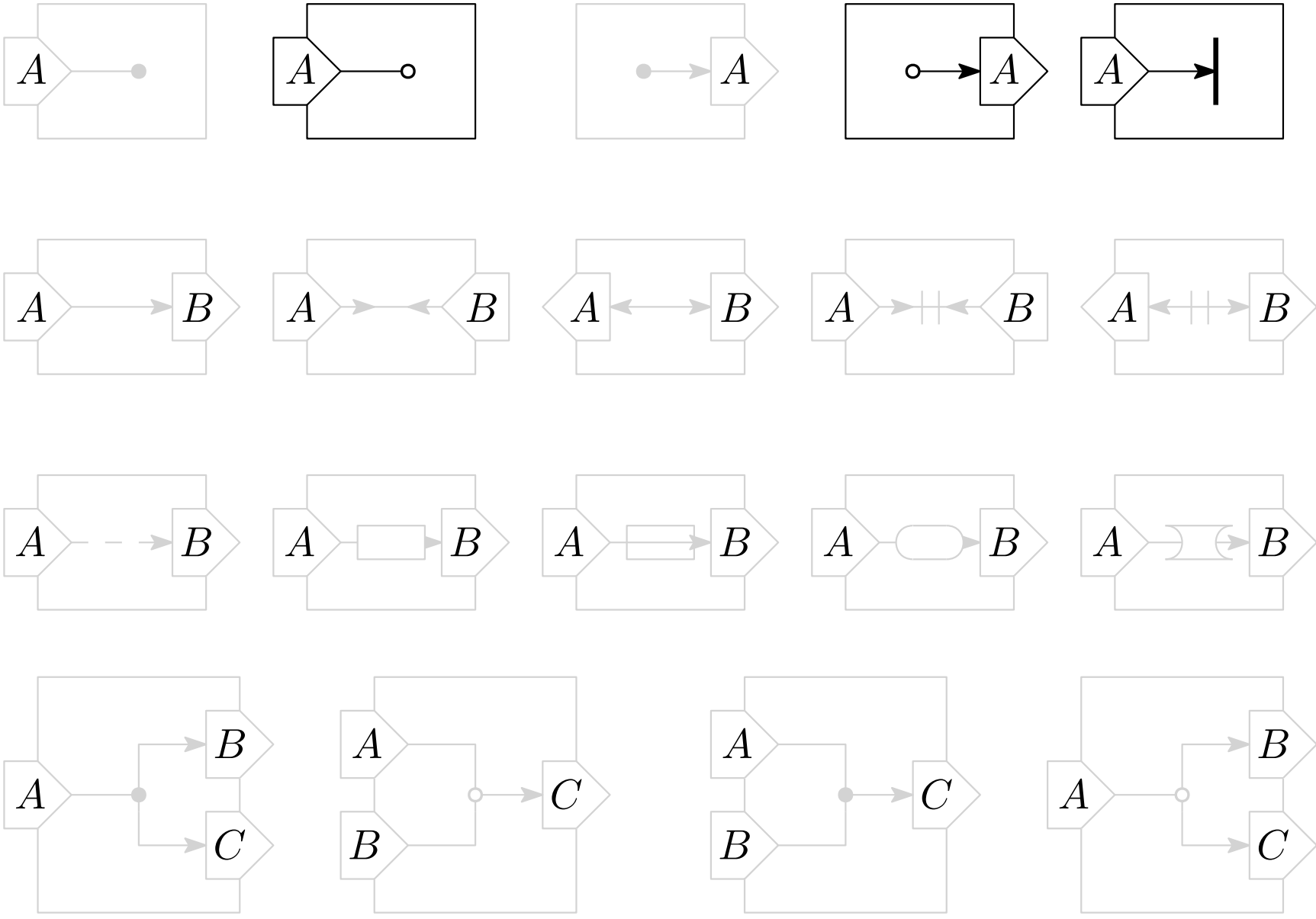
Logical property
Progress:
Always, there is a port that eventually fires

$$\forall t.(\exists s.(X_1(t+s) \neq * \vee \cdots \vee X_n(t+s) \neq *))$$

Termination:
Eventually, all ports never fire anymore

$$\exists t.(\forall s.(X_1(t+s) = * \wedge \cdots \wedge X_n(t+s) = *))$$

Termination and Progress



The End