Analysis of Concurrent Systems: Traces and Causal Structures

Łukasz Mikulski

Nicolaus Copernicus University, Faculty of Mathematics and Computer Science
Institute of Computer Science, Polish Academy of Sciences

(based on joint work with Ryszard Janicki and Jetty Kleijn and Maciej Koutny)

Toruń, 07/09/2023

Outline

Outline

Leading example

Sample program - sequence of events

- x = x+1;
- y = y+3;
- z = 2*x;
- y = y+z;
- x = x+1;

Leading example

Sample program - sequence of events

- x = x+1;
- y = y+3;
- z = 2*x;
- y = y+z;
- x = x+1;

Assumed execution semantics

For each action we first **read** all values and then, atomically, **write** it into a variable (lvalue).

Leading example

Sample program - adding action names

- x = x+1; **(a)**
- y = y+3; **(b)**
- z = 2*x; **(c)**
- y = y+z; **(d)**
- x = x+1; **(a)**

Execution (events vs actions)

We will write the execution down as **a(1)b(1)c(1)d(1)a(2)**, or equivalently as **abcda**.

Application

Problem - state space explosion

Explicit state space exploration is exposed to the exponential explosion (memory problem).

Application

Problem - state space explosion

Explicit state space exploration is exposed to the exponential explosion (memory problem).

Solution - partial order reduction

- ample sets (**dependence**, fairness, reducing the set of enabled transitions and state space)
- stubborn sets (**independence**, move stubborn transition to the prefix)
- persistent sets (**dependence**, invariants, consider only one linearisation)

Application

Problem - state space explosion

Explicit state space exploration is exposed to the exponential explosion (memory problem).

Solution - partial order reduction

- ample sets (**dependence**, fairness, reducing the set of enabled transitions and state space)
- stubborn sets (**independence**, move stubborn transition to the prefix)
- persistent sets (**dependence**, invariants, consider only one linearisation)

Practical test case

Model checking of electronic voting protocols (Selene).

Outline

Observations and observers

Event observation

- a report supplied by an observer of the system on actions executed during a run of the system

Form of a report

- action oriented description
- abstracts from state information
- distinguish between executions of the same action
- **a single run will give rise to more then one event observation (set of all such event observations – the history of this run)**

Observations and observers

Observer (sequential semantics)

- observes events with each event corresponding to the execution of an action;
- **can observe an event no more than once**;
- **may observe one event at the moment**;
- is active for a finite time;
- eventually observes only a finite number of events;
- as long as is active, it observes all events in the system;
- the order in which events are observed by an observer respects their execution order in the system;
- reports all events it observes; and
- reports events in the order they were observed.

Properties of the observation

Observation is

- denoised;
- indivisibile and instantaneous;
- atomic;
- bounded;
- finite;
- complete;
- conform;
- unabridged; and
- reorderring-free.

Leading example

Sample program

- x = x+1; **(a)**
- y = y+3; **(b)**
- z = 2*x; **(c)**
- y = y+z; **(d)**
- x = x+1; **(a)**

Equivalent executions

Single execution is a sequence of events. It is crucial to define the **dependence/independence relations**.

Leading example

Sample program

•	**(a)**	$\{x\}$
•	**(b)**	$\{y\}$
•	**(c)**	$\{x, z\}$
•	**(d)**	$\{y, z\}$
•	**(a)**	$\{x\}$

Equivalent executions

Single execution is a sequence of events. It is crucial to define the **dependence/independence relations**.

Leading example

Dependence relation

- Two actions that uses different variables are independent.
- Otherwise, they are dependent.

Independent/dependent actions

- **a** and **b** are independent, since $\{x\} \cap \{y\} = \varnothing$;
- **a** and **c** are dependent, since $\{x\} \cap \{x, z\} \neq \varnothing$;
- **a** and **d** are independent, since $\{x\} \cap \{y, z\} = \varnothing$;
- **b** and **c** are independent, since $\{y\} \cap \{x, z\} = \varnothing$;
- **b** and **d** are dependent, since $\{y\} \cap \{y, z\} \neq \varnothing$;
- **c** and **d** are dependent, since $\{x, z\} \cap \{y, z\} \neq \varnothing$;

Leading example

Sample program

- x = x+1; **(a)**
- y = y+3; **(b)**
- z = 2*x; **(c)**
- y = y+z; **(d)**
- x = x+1; **(a)**

Equivalent executions

Set of all independent pairs is $\{\langle a, b\rangle^2, \langle a, d\rangle^2, \langle b, c\rangle^2\}$.
Both **abcda** and **bacda** belong to the **history** of considered run.

Mazurkiewicz traces

Formal definition

- Alphabet of actions: $\Sigma = \{a, b, c, d\}$;
- Independence relation: $\text{ind} = \{\langle a, b\rangle^2, \langle a, d\rangle^2, \langle b, c\rangle^2\}$;
- Concurrent alphabet: $\Gamma = \langle \Sigma, \text{ind}\rangle$;
- Constructing equations: $EQ_\Gamma = \{ab = ba \mid \langle a, b\rangle \in \text{ind}\}$
- Immediate similarity: $\approx_\Gamma \subseteq \Sigma^* \times \Sigma^*$, $uabv \approx_\Gamma ubav$ for $\langle a, b\rangle \in \text{ind}$;
- Equivalence relation: $\equiv_\Gamma \subseteq \Sigma^* \times \Sigma^* = \approx_\Gamma^*$
- Traces (equivalence classes): ${}^{\Sigma^*}/_{\equiv_\Gamma}$

Trace

acbda $\approx_\Gamma$ **abcda** $\approx_\Gamma$ bacda $\approx_\Gamma$ bacad $\approx_\Gamma$ abcad $\approx_\Gamma$ acbad $\approx_\Gamma$ acabd

Alternative source of leading example

Petri net

- a triple $pn = \langle P, T, F \rangle$;
- disjoint finite sets of nodes (places P and transitions T);
- flow relation $F \subseteq (T \times P) \cup (P \times T)$

Moreover, we have the following:

- marking $M \subseteq P$;
- inputs and outputs denoted by ${}^\bullet x$ and $x^\bullet$;
- neighborhood ${}^\bullet x^\bullet = {}^\bullet x \cup x^\bullet$;
- ${}^\bullet t \neq \varnothing \neq t^\bullet$ and ${}^\bullet t \cap t^\bullet = \varnothing$.

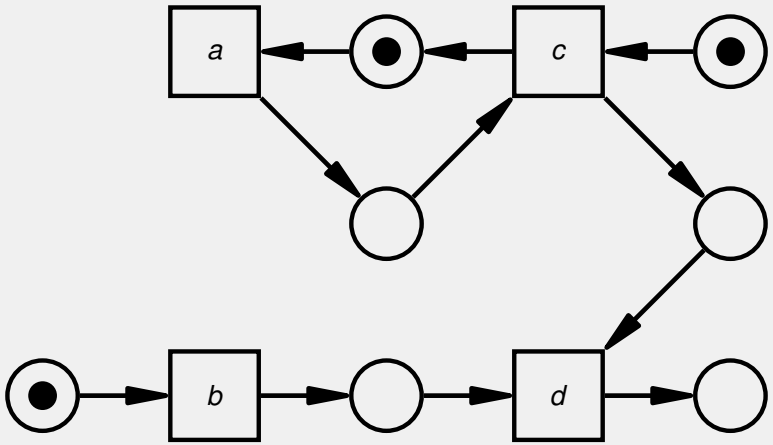
Alternative source of leading example

Elementary net system

- a tuple $en = \langle P, T, F, M_{init} \rangle$;
- $pn = \langle P, T, F \rangle$ is a net;
- $M_{init} \subseteq P$ is a nonempty initial marking;
- $t \in T$ is enabled at M ($M[t\rangle_{pn}$) if ${}^\bullet t \subseteq M \wedge t^\bullet \cap M = \varnothing$;
- $M[t\rangle_{pn} M' = M \backslash {}^\bullet t \cup t^\bullet$;
- mixed firing sequence $mfs = M_0 t_1 M_1 \ldots M_{n-1} t_n M_n \quad (n \geqslant 0)$;
- firing sequence $fs = t_1 \ldots t_n$;

Alternative source of leading example

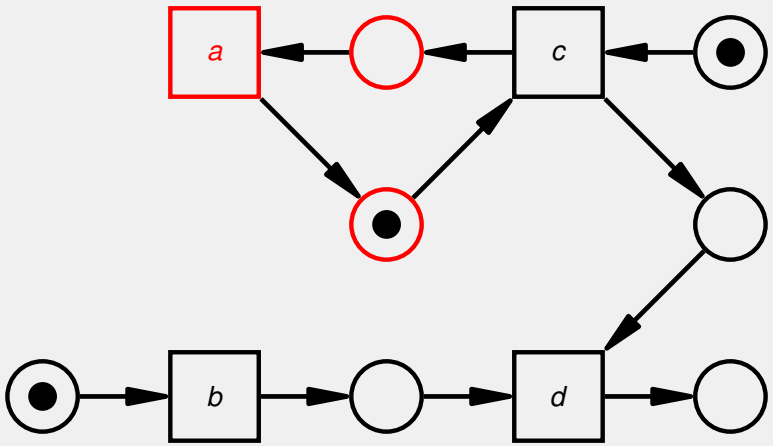
Elementary net system



λ

Alternative source of leading example

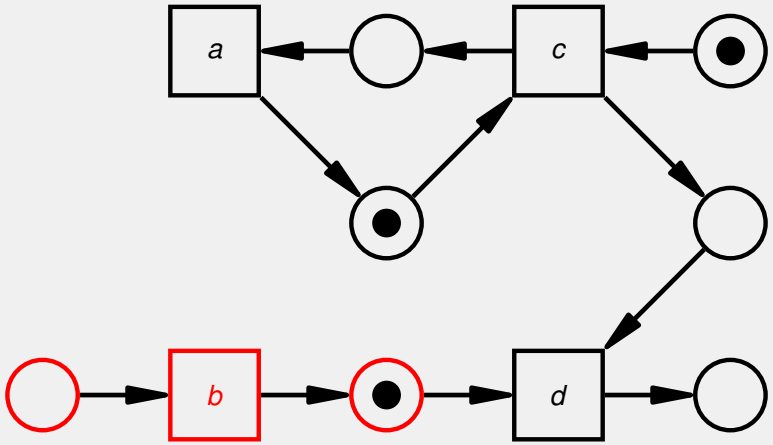
Elementary net system



a

Alternative source of leading example

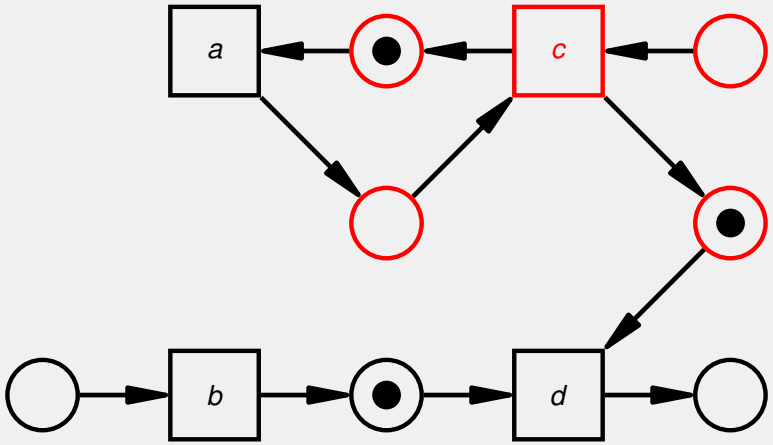
Elementary net system



ab

Alternative source of leading example

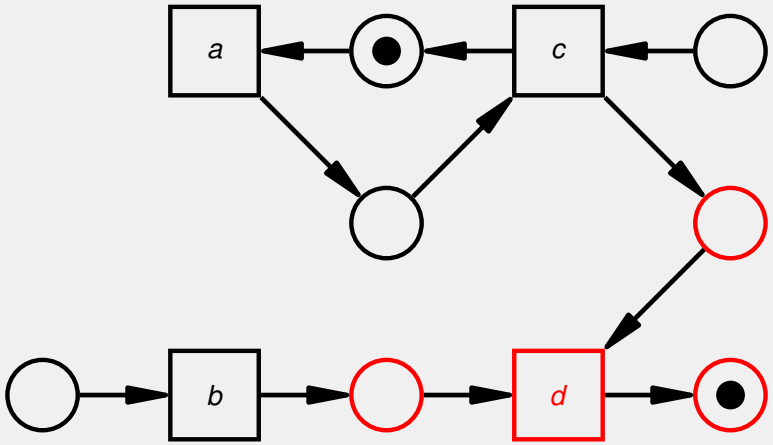
Elementary net system



abc

Alternative source of leading example

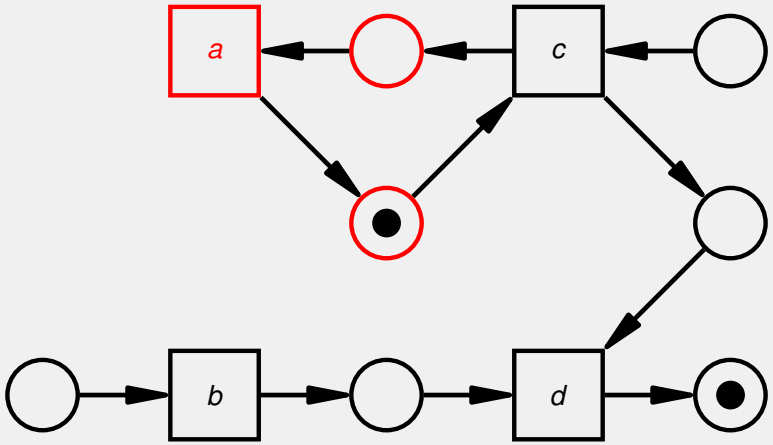
Elementary net system



abcd

Alternative source of leading example

Elementary net system



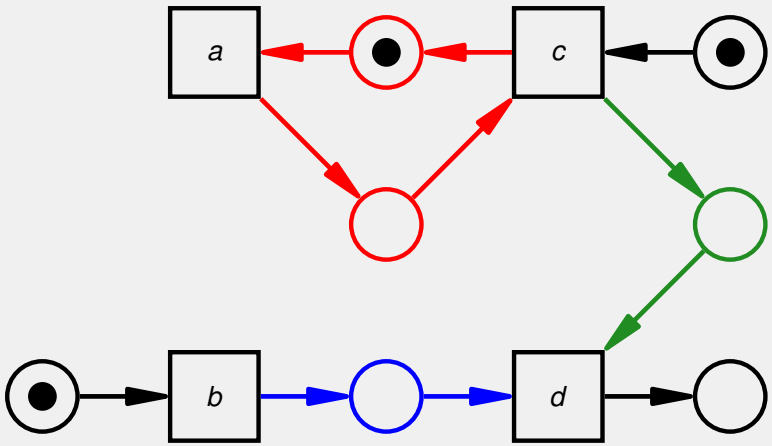
abcda

Leading example

Dependence relation

- Two actions with disjoint neighborhoods are independent.
- Otherwise, they are dependent.

Independent/dependent actions



a and **c** are dependent

c and **d** are dependent

b and **d** are dependent

Occurrence nets

Occurrence net

is a tuple $on = \langle P, T, F, \ell \rangle$ such that:

- $\langle P, T, F \rangle$ is a net with a nonempty set of places.
- $|{}^\bullet p| \leqslant 1$ and $|p^\bullet| \leqslant 1$, for every $p \in P$.
- F is acyclic

line, cut and configuration

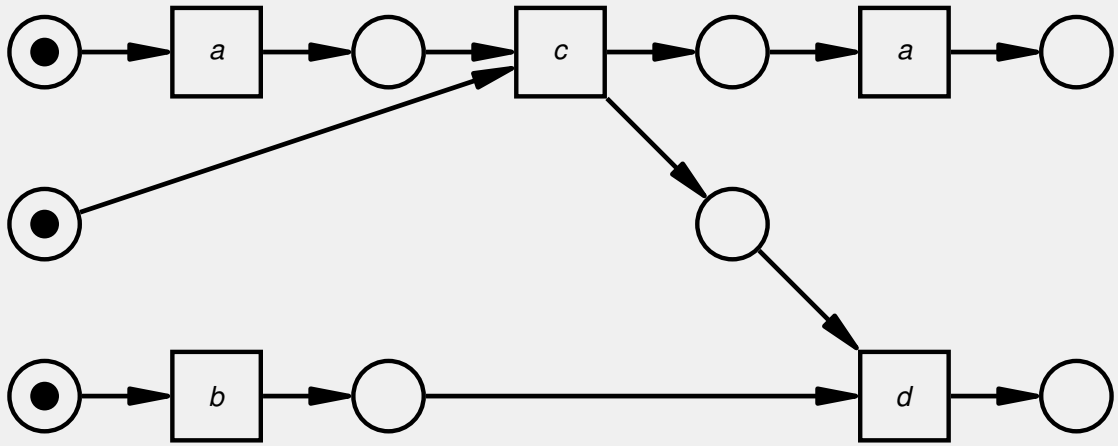
- **line** is a maximal sequence $p_0 t_1 p_1 \ldots p_{n-1} t_n p_n$ $(n \geqslant 0)$ satisfying $p_0 F t_1 F p_1 \ldots p_{n-1} F t_n F p_n$;
- **cut** is a maximal set of places *cut* such that $(cut \times cut) \cap F^+ = \varnothing$.
- **configuration** is a set of transitions *cnf* such that if $t \in cnf$ and uF^+t, then $u \in cnf$, for every $u \in T$.

Process

Possible executions

acbda $\approx_\Gamma$ **abcda** $\approx_\Gamma$ bacda $\approx_\Gamma$ bacad $\approx_\Gamma$ abcad $\approx_\Gamma$ acbad $\approx_\Gamma$ acabd

Occurrence net (unfolding)

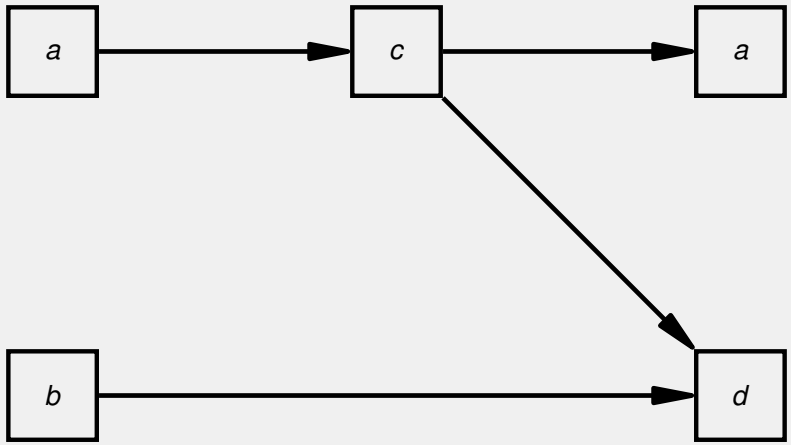


Process

Possible executions

acbda $\approx_\Gamma$ **abcda** $\approx_\Gamma$ bacda $\approx_\Gamma$ bacad $\approx_\Gamma$ abcad $\approx_\Gamma$ acbad $\approx_\Gamma$ acabd

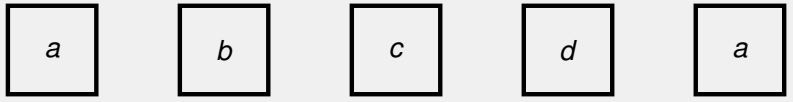
Forgetting states



Leading example

Dependence graph

- x = x+1; **(a)**
- y = y+3; **(b)**
- z = 2*x; **(c)**
- y = y+z; **(d)**
- x = x+1; **(a)**

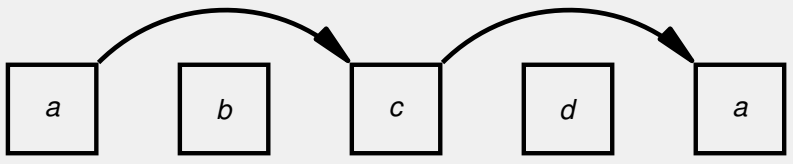


Dependence together with order of execution evolved to **causality**.

Leading example

Dependence graph

- x = x+1; **(a)**
- y = y+3; **(b)**
- z = 2*x; **(c)**
- y = y+z; **(d)**
- x = x+1; **(a)**

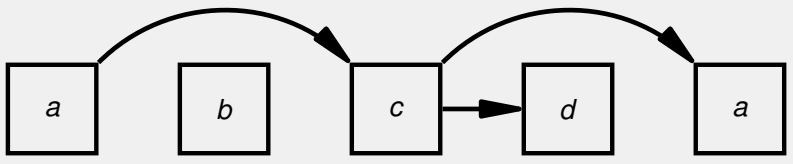


Dependence together with order of execution evolved to **causality**.

Leading example

Dependence graph

- x = x+1; **(a)**
- y = y+3; **(b)**
- z = 2*x; **(c)**
- y = y+z; **(d)**
- x = x+1; **(a)**

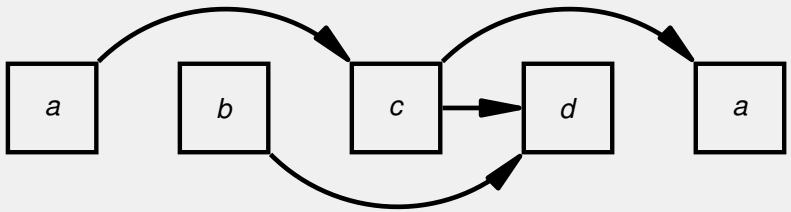


Dependence together with order of execution evolved to **causality**.

Leading example

Dependence graph

- x = x+1; **(a)**
- y = y+3; **(b)**
- z = 2*x; **(c)**
- y = y+z; **(d)**
- x = x+1; **(a)**

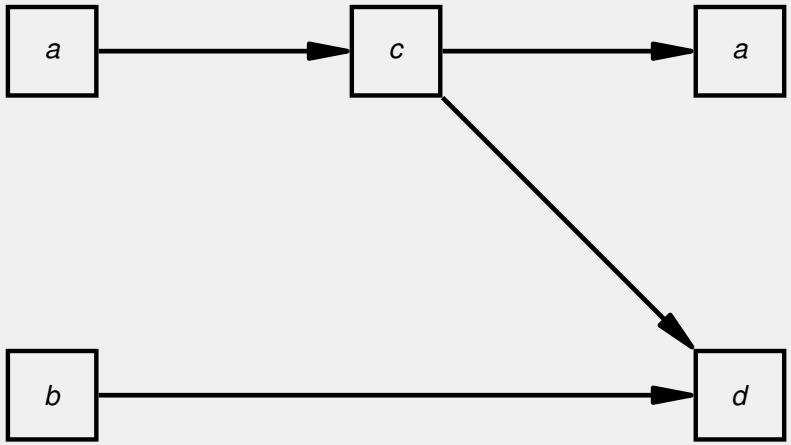


Dependence together with order of execution evolved to **causality**.

Leading example

Dependence graph

- x = x+1; **(a)**
- y = y+3; **(b)**
- z = 2*x; **(c)**
- y = y+z; **(d)**
- x = x+1; **(a)**

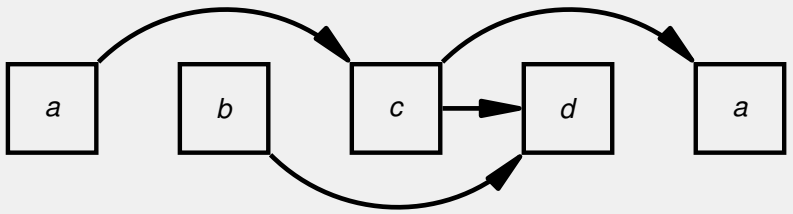


Dependence together with order of execution evolved to **causality**.

Closure

Behaviour graph

- x = x+1; **(a)**
- y = y+3; **(b)**
- z = 2*x; **(c)**
- y = y+z; **(d)**
- x = x+1; **(a)**

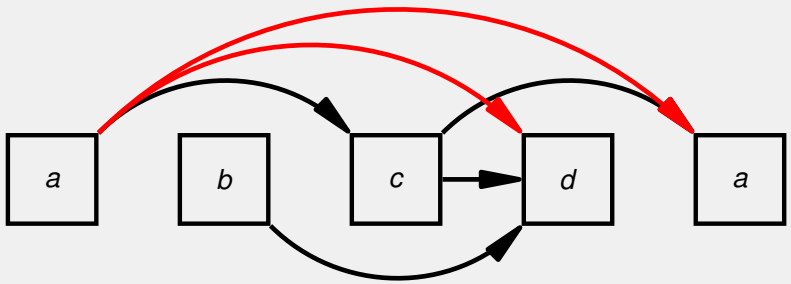


Transitive closure transforms dependence graph into **partial order**.

Closure

Behaviour graph

- x = x+1; **(a)**
- y = y+3; **(b)**
- z = 2*x; **(c)**
- y = y+z; **(d)**
- x = x+1; **(a)**

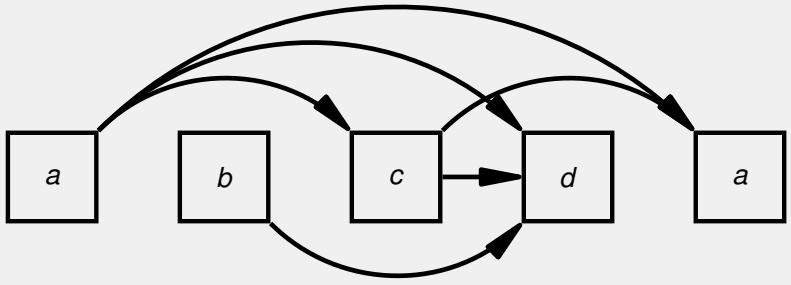


Transitive closure transforms dependence graph into **partial order**.

Saturation

Behaviour graph

- x = x+1; **(a)**
- y = y+3; **(b)**
- z = 2*x; **(c)**
- y = y+z; **(d)**
- x = x+1; **(a)**

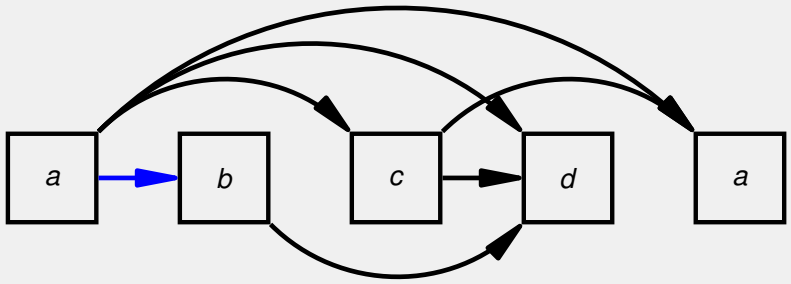


Specifying missing relationships (and closing) we **saturate** the structure.

Saturation

Behaviour graph

- x = x+1; **(a)**
- y = y+3; **(b)**
- z = 2*x; **(c)**
- y = y+z; **(d)**
- x = x+1; **(a)**

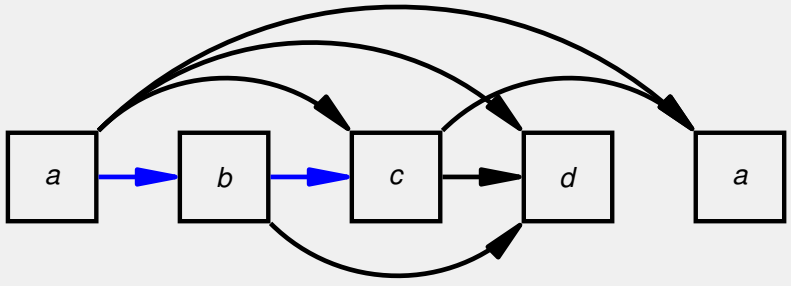


Specifying missing relationships (and closing) we **saturate** the structure.

Saturation

Behaviour graph

- x = x+1; **(a)**
- y = y+3; **(b)**
- z = 2*x; **(c)**
- y = y+z; **(d)**
- x = x+1; **(a)**

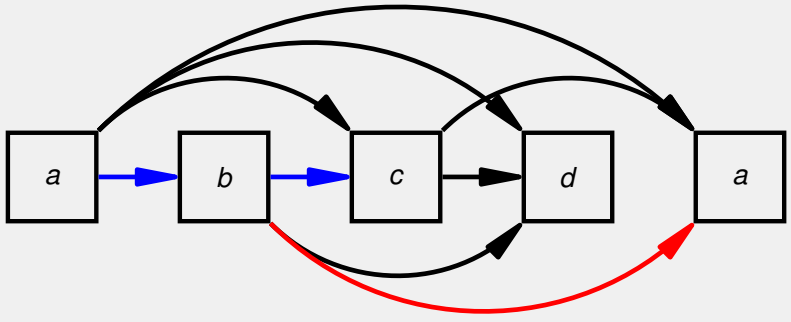


Specifying missing relationships (and closing) we **saturate** the structure.

Saturation

Behaviour graph

- x = x+1; **(a)**
- y = y+3; **(b)**
- z = 2*x; **(c)**
- y = y+z; **(d)**
- x = x+1; **(a)**

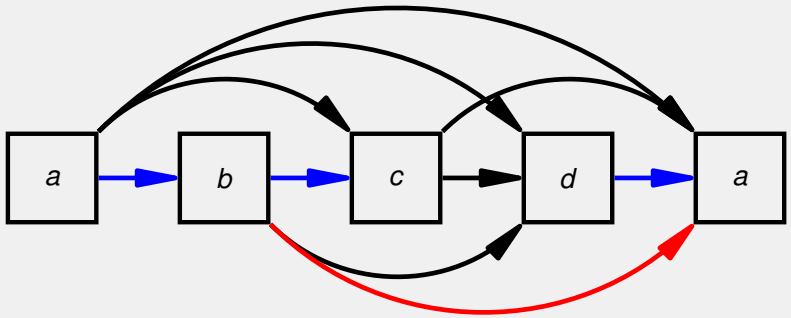


Specifying missing relationships (and closing) we **saturate** the structure.

Saturation

Behaviour graph

- x = x+1; **(a)**
- y = y+3; **(b)**
- z = 2*x; **(c)**
- y = y+z; **(d)**
- x = x+1; **(a)**



Specifying missing relationships (and closing) we **saturate** the structure.

Partial orders and total orders

Axioms for partial order

Partial order is a tuple $\langle \Delta, \prec, \ell \rangle$ such that, for all $x, y, z \in \Delta$:

$$\begin{aligned} & x \not\prec x && : \text{PO:1} \\ x \prec y \implies & x \not\prec y && : \text{PO:2} \\ x \prec y \wedge y \prec z \implies & x \prec z && : \text{PO:3} \end{aligned}$$

Axioms for total order

Total order is a tuple $\langle \Delta, \prec, \ell \rangle$ such that, for all $x, y, z \in \Delta$:

$$\begin{aligned} & x \not\prec x && : \text{TO:1} \\ x \prec y \implies & x \not\prec y && : \text{TO:2} \\ x \prec y \wedge y \prec z \implies & x \prec z && : \text{TO:3} \\ x \neq y \implies & x \prec y \wedge y \prec x && : \text{TO:4} \end{aligned}$$

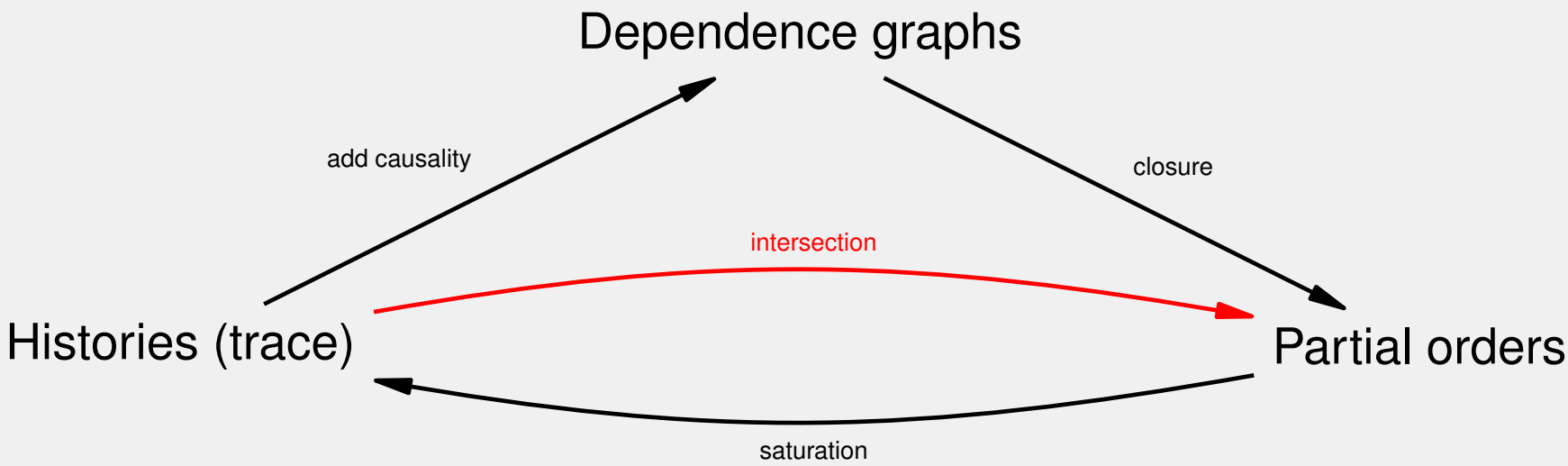
Partial orders and total orders

Summary

- Single **observation** of a run is a sequence of events (total order);
- **History** of a run is a set of possible observations;
- Order of execution together with dependence allows to construct **dependence graph** (directed acyclic graph on events);
- **Closure** of dependence graph is a partial order (order-theoretic invariant);
- **Saturation** of partial order leads to total order (any of the observations in the history) that is a **linearisation** of this partial order;

Szpilrajn theorem

Diagram



Theorem

For every partial order *po* **there exists** total order *to* containing it and *po* **is equal to the intersection** of all total orders containing it.

True concurrency paradigm

Paradigm

True concurrency paradigm (also known as 'diagonal rule' or 'diamond property') states that **simultaneity** is the same as the **possibility to occur in any order**.

Formally:

$$\begin{aligned} \forall x, y \in \Delta \quad : \quad & (\exists ob \in \mathbb{H} : \; x \frown_{ob} y) \\ & \Longleftrightarrow \\ & (\exists ob_1 \in \mathbb{H} : \; x \prec_{ob_1} y) \; \wedge \; (\exists ob_2 \in \mathbb{H} : \; y \prec_{ob_2} x) \; . \end{aligned} \tag{π_8}$$

Leading example

Equivalent executions

- abcad
- abcda
- acabd
- acbad
- acbda
- bacad
- bacda
- (ab)cad
- abc(ad)
- ac(ab)d
- acb(ad)
- a(cb)da
- bac(ad)
- (ab)cda
- (ab)c(ad)
- a(bc)(ad)

Outline

Observations and observers

Observer (sequential semantics)

- observes events with each event corresponding to the execution of an action;
- can observe an event no more than once;
- ~~may observe one event at the moment~~;
- is active for a finite time;
- eventually observes only a finite number of events;
- as long as is active, it observes all events in the system;
- the order in which events are observed by an observer respects their execution order in the system;
- reports all events it observes; and
- reports events in the order they were observed.

Observations and observers

Observer (step sequences semantics)

- observes events with each event corresponding to the execution of an action;
- can observe an event no more than once;
- **may observe more than one event at the same moment**;
- **different executions of the same action can only be observed at different moments**;
- is active for a finite time;
- eventually observes only a finite number of events;
- as long as is active, it observes all events in the system;
- the order in which events are observed by an observer respects their execution order in the system;
- reports all events it observes; and
- reports events in the order they were observed.

Properties of the observation

Observation is

- denoised;
- indivisible and instantaneous;
- **concurrent**;
- **autoconcurrency-free**;
- bounded;
- finite;
- complete;
- conform;
- unabridged; and
- reordering-free.

Leading example

Sample program - are all relationships symmetric?

- x = x+1; (a)
- y = y+3; (b)
- z = 2*x; (c)
- y = y+z; (d)
- x = x+1; (a)

ac $\not\approx$ **ca**

(ac) $\approx$ **ca**

Assumed execution semantics

For each action we first **read** all values and then, atomically, **write** it into a variable (lvalue).

New set of relations

New challenge

Dependence/independence relation is not enough. We need something which might be not symmetric.

New set of relations

New challenge

Dependence/independence relation is not enough. We need something which might be not symmetric.

First concept

We need to consider two aspects of **independence** separately:

- ~~Independence~~
- Simultaneity - actions can occur in one step
- Serialisability - occurrence in one step is equivalent with specified order

Leading example

Sample program

- x = x+1; **(a)**
- y = y+3; **(b)**
- z = 2*x; **(c)**
- y = y+z; **(d)**
- x = x+1; **(a)**

Equivalent executions

Single execution is a sequence of steps. It is crucial to define **valid steps** (simultaneity relation) and **serialisability relation**.

Leading example

Simultaneity relation

- Two actions can occur in the same step if they change different variables.
- Otherwise, they cannot.

Simultaneous actions

- **a** and **b** are simultaneous, since $x \neq y$;
- **a** and **c** are simultaneous, since $x \neq z$;
- **a** and **d** are simultaneous, since $x \neq y$;
- **b** and **c** are simultaneous, since $y \neq z$;
- **b** and **d** are not simultaneous, since they both change y;
- **c** and **d** are simultaneous, since $z \neq y$;

Leading example

Serialisability relation

- Two actions **a** and **b** occurring in the same step can be serialised as **ab** if **a** changes variable not used by **b**.
- Otherwise, they cannot.

Serialisable actions

- **(ab)** can be serialised as **ab** or **ba**, since **a** and **b** are independent;
- **(ac)** can be serialised only as **ca**, since **a** changes value of x;
- **(ad)** can be serialised as **ad** or **da**, since **a** and **d** are independent;
- **(cb)** can be serialised as **cb** or **bc**, since **b** and **c** are independent;
- **(bd)** cannot be serialised, since **b** and **d** are not simultaneous;
- **(cd)** can be serialised only as **dc**, since **c** changes value of z;

Leading example

Sample program

- x = x+1; **(a)**
- y = y+3; **(b)**
- z = 2*x; **(c)**
- y = y+z; **(d)**
- x = x+1; **(a)**

Equivalent executions

Set of all simultaneous pairs is $\{\langle a, b\rangle^2, \langle a, c\rangle^2, \langle a, d\rangle^2, \langle b, c\rangle^2, \langle c, d\rangle^2\}$, while for serialisability we have $\{\langle a, b\rangle^2, \langle c, a\rangle, \langle a, d\rangle^2, \langle b, c\rangle^2, \langle d, c\rangle\}$.

Combined traces

Formal definition

- Alphabet of actions: $\Sigma = \{a, b, c, d\}$;
- Simultaneity relation: $\text{sim} = \{\langle a, b\rangle^2, \langle a, c\rangle^2, \langle a, d\rangle^2, \langle b, c\rangle^2, \langle c, d\rangle^2\}$;
- Serialisability relation: $\text{ser} = \{\langle a, b\rangle^2, \langle c, a\rangle, \langle a, d\rangle^2, \langle b, c\rangle^2, \langle d, c\rangle\}$;
- Concurrent alphabet: $\Theta = \langle \Sigma, \text{sim}, \text{ser}\rangle$;
- Valid steps: $S_\Theta = \{A \subseteq \Sigma \mid A \neq \varnothing \wedge (A \times A) \backslash id_\Sigma \subseteq \text{sim}\}$
- Constructing equations: $EQ_\Theta = \{AB = A \cup B \mid A \times B \subseteq \text{ser}\}$
- Immediate similarity:
$$\approx_\Theta \subseteq S_\Theta^* \times S_\Theta^*,\ uABv \approx_\Theta u(A \cup B)v \text{ for } A \times B \subseteq \text{ser};$$
- Equivalence relation: $\equiv_\Theta \subseteq S_\Theta^* \times S_\Theta^* = \approx_\Theta^*$
- Traces (equivalence classes): $S_\Theta^* / \equiv_\Theta$

Leading example

Equivalent executions

- ab**ca**d
- abcda
- a**ca**bd
- acbad
- acbda
- ba**ca**d
- bacda

- (ab)**ca**d
- abc(ad)
- ac(ab)d
- acb(ad)
- a(cb)da
- bac(ad)
- (ab)cda
- (ab)c(ad)
- a(bc)(ad)

- ab(ca)d
- a(ca)bd
- ba(ca)d
- (ab)(ca)d
- a(cab)d

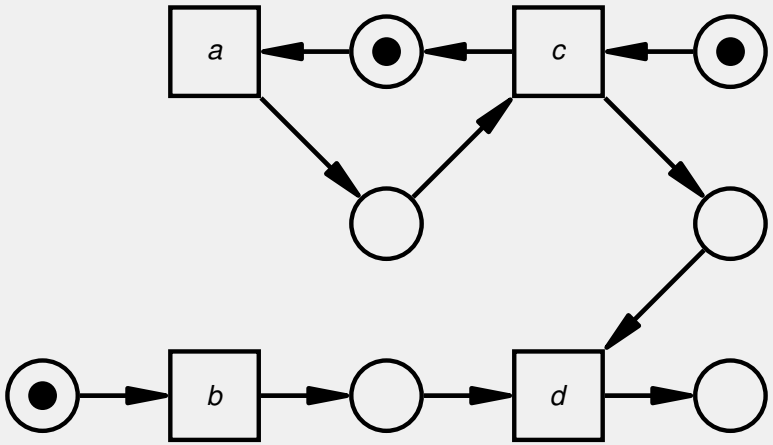
System model

Elementary net system with activators

- a tuple $en = \langle P, T, Act, F, M_{init} \rangle$;
- $pn = \langle P, T, F \rangle$ is a net;
- $Act \subseteq P \times T$ is a set of activation arcs;
- ${}^{\circledcirc}t = \{p \in P \mid \langle p, t \rangle \in Act\}$ are the **activator places** of transition t;
- $M_{init} \subseteq P$ is a nonempty initial marking;
- $t \in T$ is enabled at M ($M[t\rangle_{pn}$) if ${}^{\bullet}t \cup {}^{\circledcirc}t \subseteq M \wedge t^{\bullet} \cap M = \varnothing$;
- $M[t\rangle_{pn} M' = M \backslash {}^{\bullet}t \cup t^{\bullet}$;
- $U \subseteq T$ is enabled at M ($M[U\rangle_{pn}$) if $\forall_{t \in U}(M[t\rangle_{pn} \wedge \forall_{t \neq t' \in U} {}^{\bullet}t \cap {}^{\bullet}t' = \varnothing)$;
- mixed firing sequence $mfs = M_0 U_1 M_1 \ldots M_{n-1} U_n M_n \quad (n \geqslant 0)$;
- firing sequence $fs = U_1 \ldots U_n$;

Alternative source of leading example?

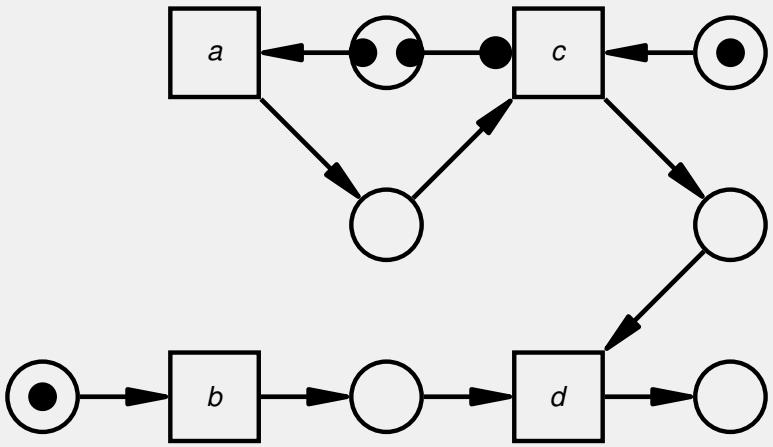
Elementary net system



λ

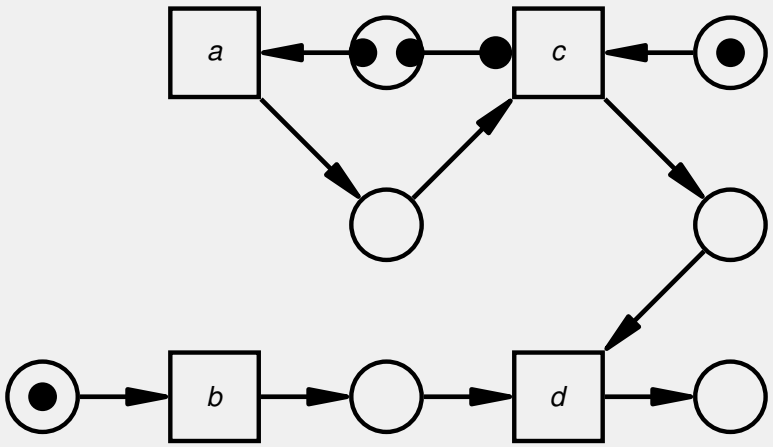
Alternative source of leading example?

Non-safe solution



Alternative source of leading example?

Non-safe solution

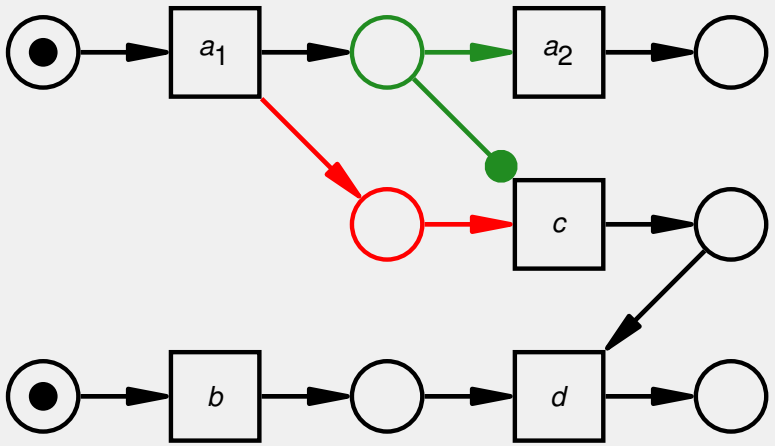


Problem

In elementary systems, we can **never** execute the same transition **in two consecutive steps**!

Label splitting

Elementary net system with activators



a_1 and **c** are not simultaneous

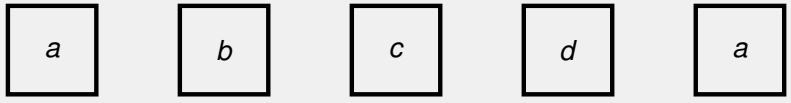
a_2 and **c** are not serialisable

c and **a_2** are serialisable

Leading example

Dependence graph

- x = x+1; **(a)**
- y = y+3; **(b)**
- z = 2*x; **(c)**
- y = y+z; **(d)**
- x = x+1; **(a)**

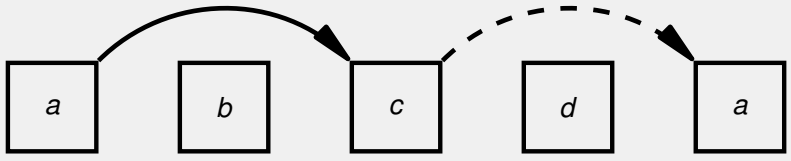


Serialisability together with order of execution is **weak causality**.
Weak causality is **'not later than'** (earlier or at the same time) relation.
Lack of serialisability together with order of execution is **causality**.

Leading example

Dependence graph

- x = x+1; **(a)**
- y = y+3; **(b)**
- z = 2*x; **(c)**
- y = y+z; **(d)**
- x = x+1; **(a)**

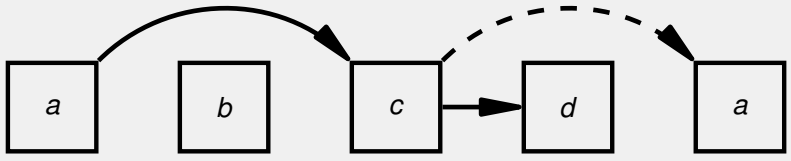


Serialisability together with order of execution is **weak causality**.
Weak causality is **'not later than'** (earlier or at the same time) relation.
Lack of serialisability together with order of execution is **causality**.

Leading example

Dependence graph

- x = x+1; **(a)**
- y = y+3; **(b)**
- z = 2*x; **(c)**
- y = y+z; **(d)**
- x = x+1; **(a)**

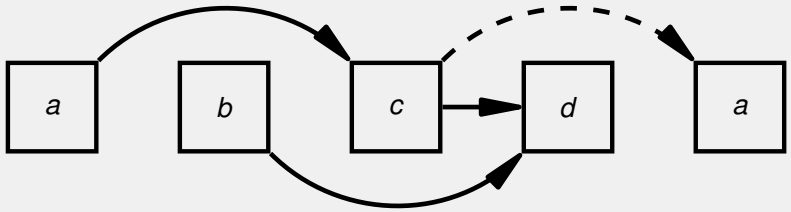


Serialisability together with order of execution is **weak causality**.
Weak causality is **'not later than'** (earlier or at the same time) relation.
Lack of serialisability together with order of execution is **causality**.

Leading example

Dependence graph

- x = x+1; **(a)**
- y = y+3; **(b)**
- z = 2*x; **(c)**
- y = y+z; **(d)**
- x = x+1; **(a)**

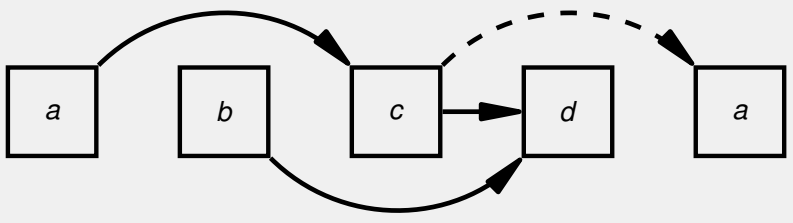


Serialisability together with order of execution is **weak causality**.
Weak causality is **'not later than'** (earlier or at the same time) relation.
Lack of serialisability together with order of execution is **causality**.

Closure

Relational structure

- x = x+1; **(a)**
- y = y+3; **(b)**
- z = 2*x; **(c)**
- y = y+z; **(d)**
- x = x+1; **(a)**

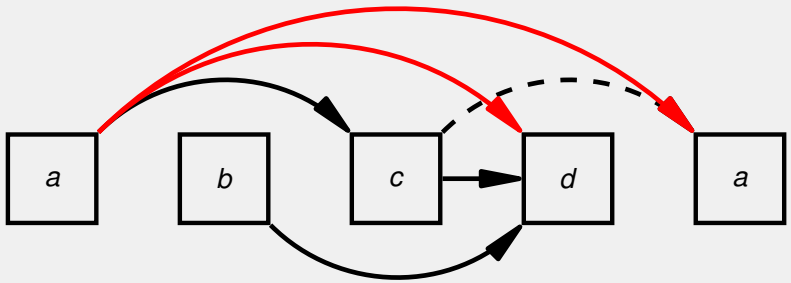


Many types of transitivity - two **causality** arcs, two **weak causality** arcs and **mixed** situations. Intuition works fine.

Closure

Relational structure

- x = x+1; **(a)**
- y = y+3; **(b)**
- z = 2*x; **(c)**
- y = y+z; **(d)**
- x = x+1; **(a)**

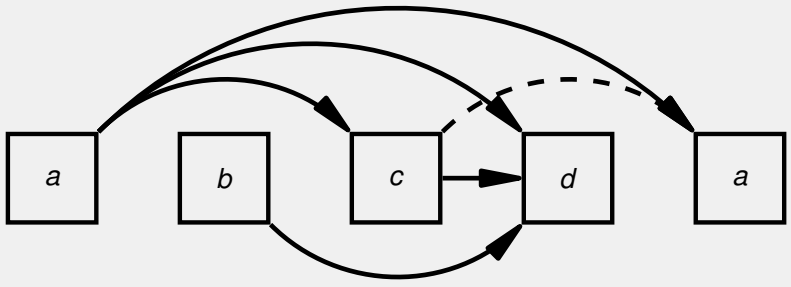


Many types of transitivity - two **causality** arcs, two **weak causality** arcs and **mixed** situations. Intuition works fine.

Saturation

Relational structure

- x = x+1; **(a)**
- y = y+3; **(b)**
- z = 2*x; **(c)**
- y = y+z; **(d)**
- x = x+1; **(a)**

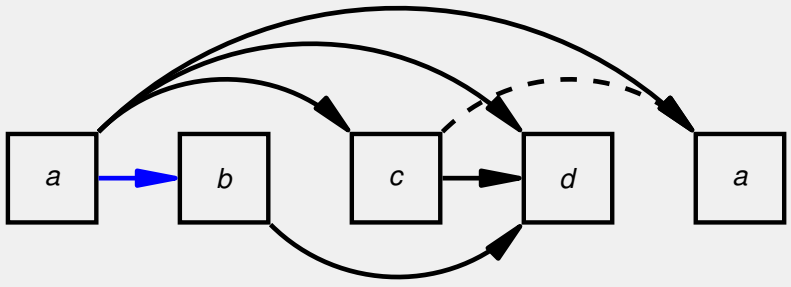


We can **specify** either new causality or new weak causality arcs (and **close** the structure).

Saturation

Relational structure

- x = x+1; **(a)**
- y = y+3; **(b)**
- z = 2*x; **(c)**
- y = y+z; **(d)**
- x = x+1; **(a)**

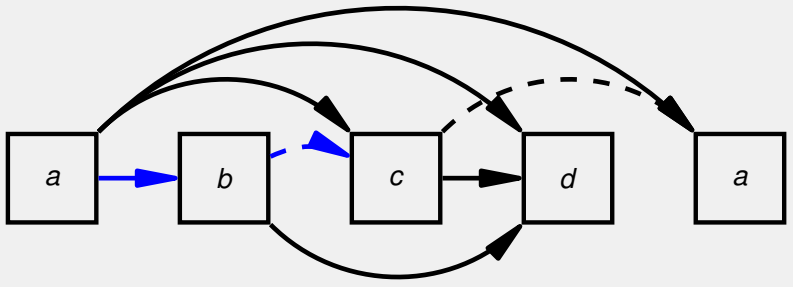


We can **specify** either new causality or new weak causality arcs (and **close** the structure).

Saturation

Relational structure

- x = x+1; **(a)**
- y = y+3; **(b)**
- z = 2*x; **(c)**
- y = y+z; **(d)**
- x = x+1; **(a)**

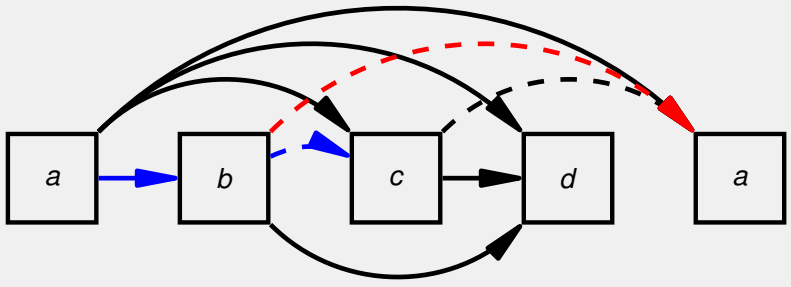


We can **specify** either new causality or new weak causality arcs (and **close** the structure).

Saturation

Relational structure

- x = x+1; **(a)**
- y = y+3; **(b)**
- z = 2*x; **(c)**
- y = y+z; **(d)**
- x = x+1; **(a)**

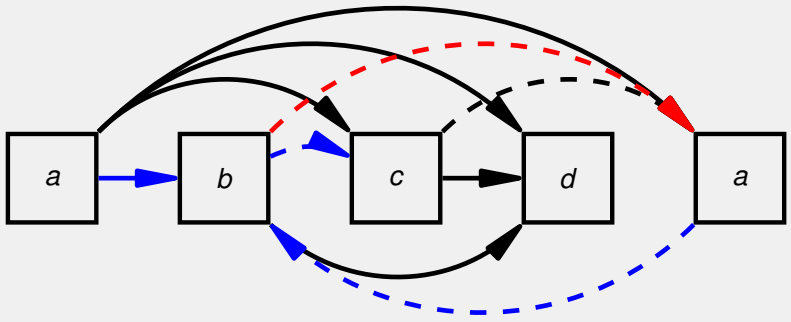


We can **specify** either new causality or new weak causality arcs (and **close** the structure).

Saturation

Relational structure

- x = x+1; **(a)**
- y = y+3; **(b)**
- z = 2*x; **(c)**
- y = y+z; **(d)**
- x = x+1; **(a)**

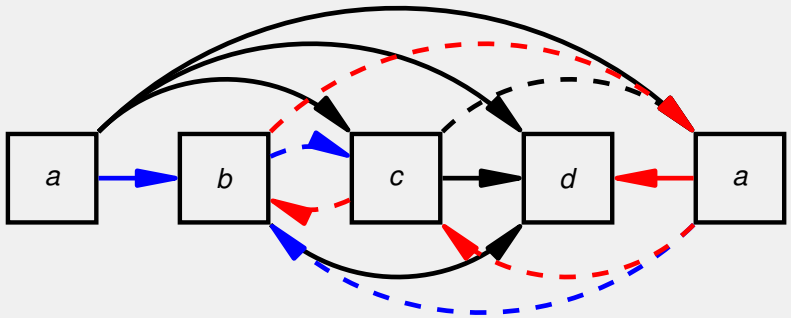


We can **specify** either new causality or new weak causality arcs (and **close** the structure).

Saturation

Relational structure

- x = x+1; **(a)**
- y = y+3; **(b)**
- z = 2*x; **(c)**
- y = y+z; **(d)**
- x = x+1; **(a)**

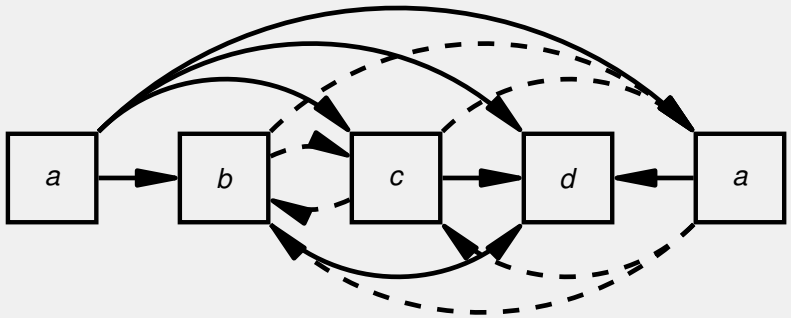


We can **specify** either new causality or new weak causality arcs (and **close** the structure).

Saturation

Relational structure

- x = x+1; **(a)**
- y = y+3; **(b)**
- z = 2*x; **(c)**
- y = y+z; **(d)**
- x = x+1; **(a)**



Obtained step sequence is:

a(abc)d.

Closed structures

Axioms for stratified order structures

stratified order structure is a ralational structure $\langle \Delta, \prec, \sqsubset, \ell \rangle$ such that, for all $x, y, z \in \Delta$:

$$
\begin{array}{rcrcll}
 & & x \not\sqsubset x & : & \text{SO:1} \\
 & x \prec y & \Longrightarrow & x \sqsubset y & : & \text{SO:2} \\
x \neq z \quad \wedge & x \sqsubset y \sqsubset z & \Longrightarrow & x \sqsubset z & : & \text{SO:3} \\
x \sqsubset y \prec z \quad \vee & x \prec y \sqsubset z & \Longrightarrow & x \prec z & : & \text{SO:4}
\end{array}
$$

Closure

$$\langle \Delta, \prec, \sqsubset, \ell \rangle \mapsto \langle \Delta, \mathrm{pre}_{\mathrm{cos}}(\prec, \sqsubset), (\prec \cup \sqsubset)^{\curlywedge}, \ell \rangle$$

where

- $\mathrm{pre}_{\mathrm{cos}}(Q, R) = (Q \cup R)^* \circ Q \circ (Q \cup R)^*$
- $R^{\curlywedge} = R^+ \backslash R^0$.

Maximal structures

Axioms for layered concurrent structures

Layered concurrent structure is a relational structure $\langle \Delta, \prec, \sqsubset, \ell \rangle$ such that, for all $x, y, z \in \Delta$:

$$
\begin{array}{lcl}
 & x \not\sqsubset x & : \ \textsc{lc}{:}1 \\
x \prec y \implies & x \sqsubset y \quad \wedge \ y \not\sqsubset x & : \ \textsc{lc}{:}2 \\
x \prec y \implies & x \prec z \quad \vee \ z \prec y & : \ \textsc{lc}{:}3 \\
x \neq y \implies & x \sqsubset y \sqsubset x \ \vee \ x \prec^{sym} y & : \ \textsc{lc}{:}4
\end{array}
$$

Equivalent objects

Layered concurrent structures correspond to **step sequences** and **stratified partial orders** (partial orders where $(\Delta \times \Delta) \setminus \prec^{sym}$ is an equivalence relation).

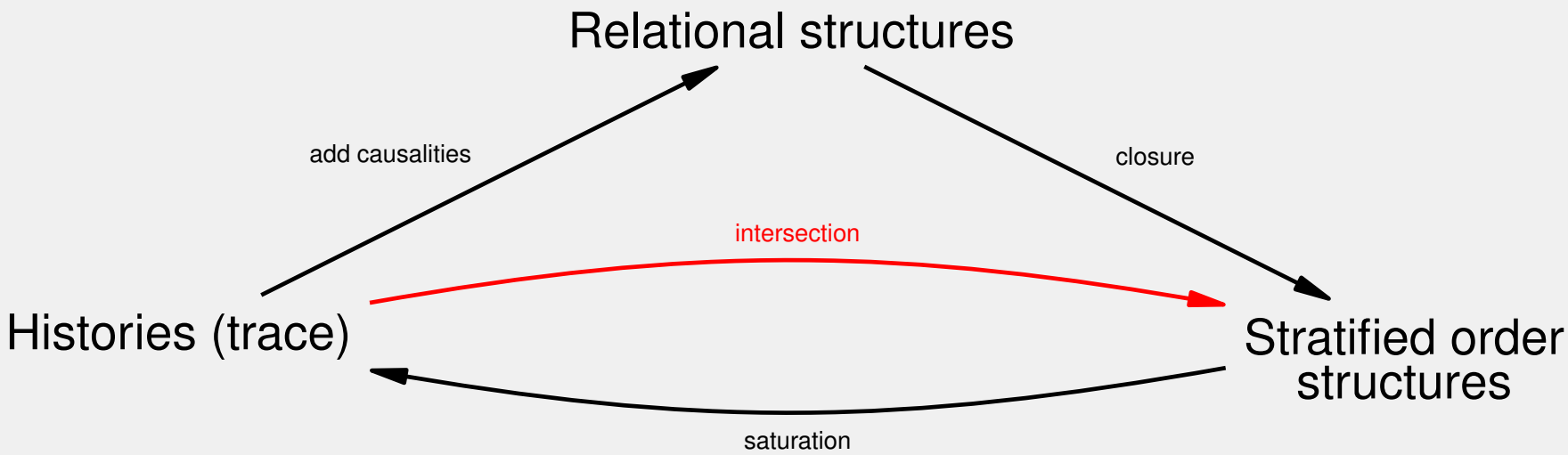
Stratified order and layered concurrent structures

Summary

- Single **observation** of a run is a step sequence of events (stratified order);
- **History** of a run is a set of possible observations;
- Order of execution together with simultaneity and serialisability allows to construct **dependence graph** (relational structure on events);
- **Closure** of dependence graph is a stratified order structure (order-theoretic invariant);
- **Saturation** of stratified order structure can be extended to layered concurrent structure (any of the observations in the history);

Szpilrajn theorem

Diagram



Theorem

For every stratified order structure *sos* **there exists** layered concurrent structure *los* containing it and *sos* **is equal to the intersection** of layered concurrent structures containing it.

Less demanding paradigm

Paradigm

To express 'not later than' relations we can weaken true concurrency paradigm. The less demanding formulation states that **possibility to occur in any order** implies **simultaneity**.

Formally:

$$\forall x, y \in \Delta \quad : \quad (\exists po \in \mathbb{H} : \; x \frown_{po} y)$$

$$\Longleftarrow \qquad (\pi_3)$$

$$(\exists po \in \mathbb{H} : \; x \prec_{po} y) \; \wedge \; (\exists po \in \mathbb{H} : \; y \prec_{po} x) \; .$$

Leading example

Sample program - is it possible to commute events that are cannot occur symultaneously?

- x = x+1; (a)
- y = y+3; (b)
- z = 2*x; (c)
- y = y+z; (d)
- x = x+1; (a)

bd $\approx$ db

(bd) indefinite

Assumed execution semantics

For each action we first **read** all values and then, atomically, **write** it into a variable (lvalue).

Even larger set of relations

New challenge

Simultaneity and serialisability relations are not enough. We need something new to capture interleaving.

Even larger set of relations

New challenge

Simultaneity and serialisability relations are not enough. We need something new to capture interleaving.

First attempt

We can add **interleaving** not related (or even disjoint) with simultaneity as third relation.

Even larger set of relations

New challenge

Simultaneity and serialisability relations are not enough. We need something new to capture interleaving.

First attempt

We can add **interleaving** not related (or even disjoint) with simultaneity as third relation.

Economical solution

Let us drop the assumption that serialisability is included in simultaneity:

- Simultaneity - actions can occur in one step
- ~~Serialisability~~
- Sequentialisability - serialisability enriched by interleaving

Leading example

Sequentialisability relation

- Two actions **a** and **b** can be sequentialised if either **a** changes variable not used by **b** or they both changes the same variable with the same eventual results for **ab** and **ba**;
- Otherwise, they cannot.

Serialisable actions

- **(ab)** can be sequentialised as **ab** or **ba** (**a** and **b** are independent);
- **(ac)** can be sequentialised only as **ca** (they are serialisable this way);
- **(ad)** can be sequentialised as **ad** or **da** (**a** and **d** are independent);
- **(cb)** can be sequentialised as **cb** or **bc** (**b** and **c** are independent);
- **b** and **d** are sequentialisable, since **b** and **d** are not simultaneous but results of **(bd)** and **(db)** are the same;
- **(cd)** can be sequentialised only as **dc** (they are serialisable this way).

Leading example

Sample program

- x = x+1; **(a)**
- y = y+3; **(b)**
- z = 2*x; **(c)**
- y = y+z; **(d)**
- x = x+1; **(a)**

Equivalent executions

Set of all simultaneous pairs is $\{\langle a, b\rangle^2, \langle a, c\rangle^2, \langle a, d\rangle^2, \langle b, c\rangle^2, \langle c, d\rangle^2\}$, while sequentialisability: $\{\langle a, b\rangle^2, \langle c, a\rangle, \langle a, d\rangle^2, \langle b, c\rangle^2, \langle b, d\rangle^2, \langle d, c\rangle\}$.

Step traces

Formal definition

- Alphabet of actions: $\Sigma = \{a, b, c, d\}$;
- Simultaneity relation: $\text{sim} = \{\langle a, b\rangle^2, \langle a, c\rangle^2, \langle a, d\rangle^2, \langle b, c\rangle^2, \langle c, d\rangle^2\}$;
- Sequentialisability relation:
$$\text{seq} = \{\langle a, b\rangle^2, \langle c, a\rangle, \langle a, d\rangle^2, \langle b, c\rangle^2, \langle b, d\rangle^2, \langle d, c\rangle\};$$
- Concurrent alphabet: $\Theta = \langle \Sigma, \text{sim}, \text{seq}\rangle$;
- Valid steps: $S_\Theta = \{A \subseteq \Sigma \mid A \neq \varnothing \wedge (A \times A) \backslash id_\Sigma \subseteq \text{sim}\}$
- Constructing equations: $EQ_\Theta = \{AB = A \cup B \mid A \times B \subseteq \text{seq} \cap \text{sim}$
$$AB = BA \mid A \times B \subseteq \text{seq} \cap \text{seq}^{-1}\}$$
- Immediate similarity: $\approx_\Theta \subseteq S_\Theta^* \times S_\Theta^*$,
$$uABv \approx_\Theta u(A \cup B)v \text{ for } A \times B \subseteq \text{seq} \cap \text{sim};$$
$$\text{or } uABv \approx_\Theta uBAv \text{ for } A \times B \subseteq \text{seq} \cap \text{seq}^{-1};$$
- Equivalence relation: $\equiv_\Theta \subseteq S_\Theta^* \times S_\Theta^* = \approx_\Theta^*$
- Traces (equivalence classes): $S_\Theta^* / \equiv_\Theta$

Leading example

Equivalent executions

- abcad
- abcda
- aca**bd**
- acbad
- ac**bd**a
- bacad
- bacda

- (ab)cad
- abc(ad)
- ac**(ab)d**
- ac**b(ad)**
- a(cb)da
- bac(ad)
- (ab)cda
- (ab)c(ad)
- a(bc)(ad)

- ab(ca)d
- a(ca)**bd**
- ba(ca)d
- (ab)(ca)d
- a(cab)d

- aca**db**
- ac**db**a
- ac**d(ab)**
- ac**(ad)b**
- a(ca)**db**

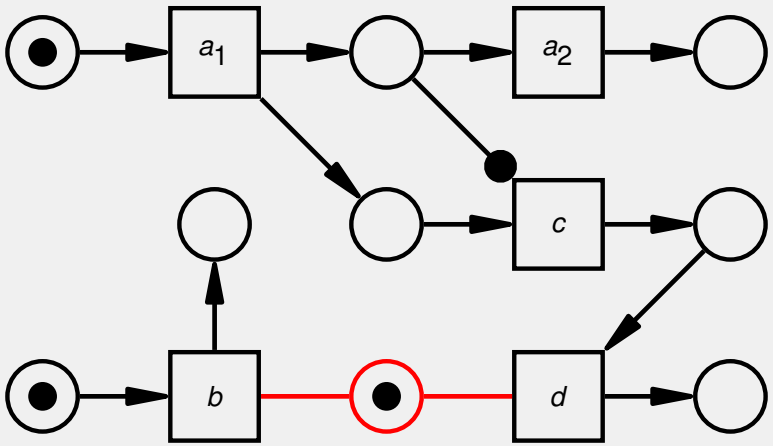
System model

Elementary net system with activators and mutexes

- a tuple $en = \langle P, T, Act, Mut, F, M_{init} \rangle$;
- $pn = \langle P, T, F \rangle$ is a net, Act is a set of activation arcs;
- $Mut \subseteq P \times T$ is a set of mutex arcs;
- ${}^{\odot}t = \{p \in P \mid \langle p, t \rangle \in Mut\}$ are the **mutex places** of transition t;
- $M_{init} \subseteq P$ is a nonempty initial marking;
- $t \in T$ is enabled at M ($M[t\rangle_{pn}$) if ${}^{\bullet}t \cup {}^{\circledcirc}t \cup {}^{\odot}t \subseteq M \wedge t^{\bullet} \cap M = \varnothing$;
- $M[t\rangle_{pn} M' = M \backslash {}^{\bullet}t \cup t^{\bullet}$;
- $U \subseteq T$ is enabled at M ($M[U\rangle_{pn}$) if
$$\forall_{t \in U}(M[t\rangle_{pn} \wedge \forall_{t \neq t' \in U}({}^{\bullet}t \cup {}^{\odot}t) \cap ({}^{\bullet}t' \cup {}^{\odot}t') = \varnothing);$$
- mixed firing sequence $mfs = M_0 U_1 M_1 \ldots M_{n-1} U_n M_n \quad (n \geqslant 0)$;
- firing sequence $fs = U_1 \ldots U_n$;

Label splitting again

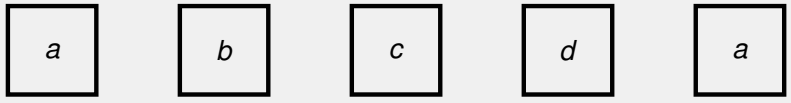
Elementary net system with activators and mutexes



Leading example

Dependence graph

- x = x+1; **(a)**
- y = y+3; **(b)**
- z = 2*x; **(c)**
- y = y+z; **(d)**
- x = x+1; **(a)**

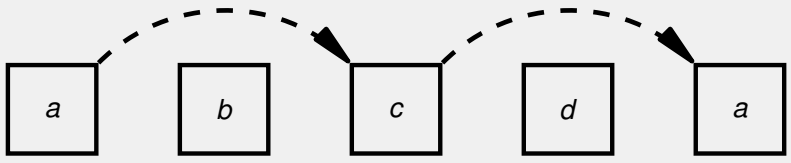


Sequentialization together with order of execution is **weak causality**.
Mutex is related with the **lack of simultaneity or sequentializability**.
Causality is the intersection of mutex and weak causality.

Leading example

Dependence graph

- x = x+1; **(a)**
- y = y+3; **(b)**
- z = 2*x; **(c)**
- y = y+z; **(d)**
- x = x+1; **(a)**

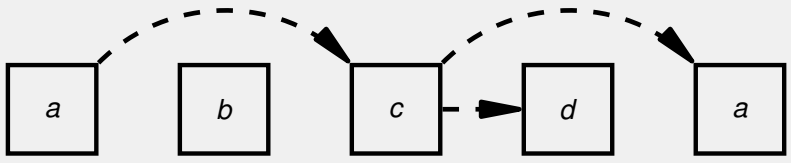


Sequentializability together with order of execution is **weak causality**. **Mutex** is related with the **lack of simultaneity or sequentializability**. **Causality** is the intersection of mutex and weak causality.

Leading example

Dependence graph

- x = x+1; **(a)**
- y = y+3; **(b)**
- z = 2*x; **(c)**
- y = y+z; **(d)**
- x = x+1; **(a)**

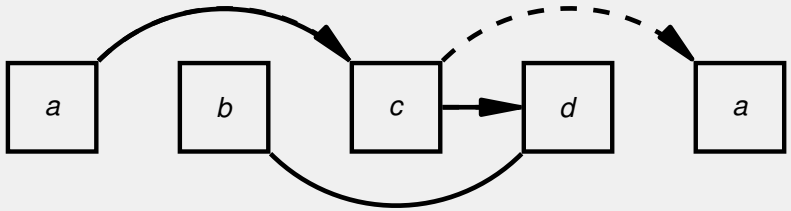


Sequentializability together with order of execution is **weak causality**. **Mutex** is related with the **lack of simultaneity or sequentializability**. **Causality** is the intersection of mutex and weak causality.

Leading example

Dependence graph

- x = x+1; **(a)**
- y = y+3; **(b)**
- z = 2*x; **(c)**
- y = y+z; **(d)**
- x = x+1; **(a)**

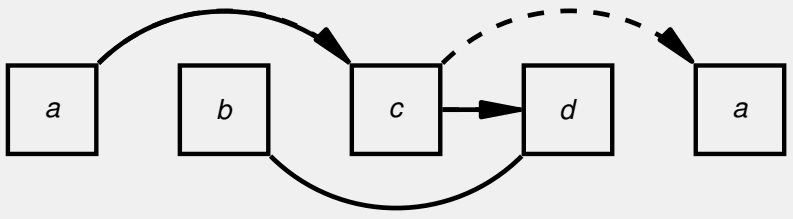


Sequentializability together with order of execution is **weak causality**.
Mutex is related with the **lack of simultaneity or sequentializability**.
Causality is the intersection of mutex and weak causality.

Closure

Relational structure

- x = x+1; **(a)**
- y = y+3; **(b)**
- z = 2*x; **(c)**
- y = y+z; **(d)**
- x = x+1; **(a)**

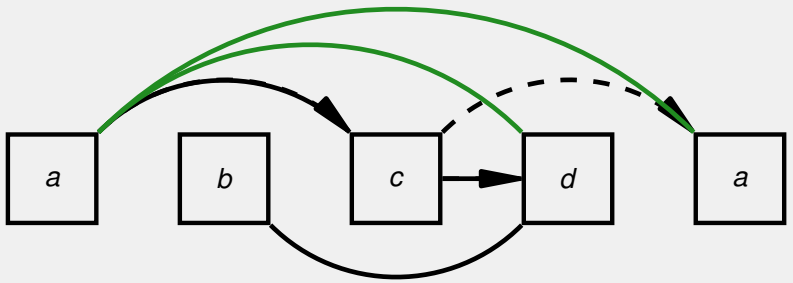


We need to complete induced **mutex edges** and **weak causality arcs**.

Closure

Relational structure

- x = x+1; **(a)**
- y = y+3; **(b)**
- z = 2*x; **(c)**
- y = y+z; **(d)**
- x = x+1; **(a)**

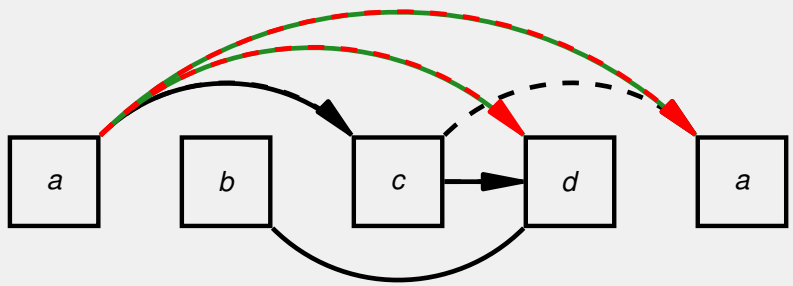


We need to complete induced **mutex edges** and **weak causality arcs**.

Closure

Relational structure

- x = x+1; **(a)**
- y = y+3; **(b)**
- z = 2*x; **(c)**
- y = y+z; **(d)**
- x = x+1; **(a)**

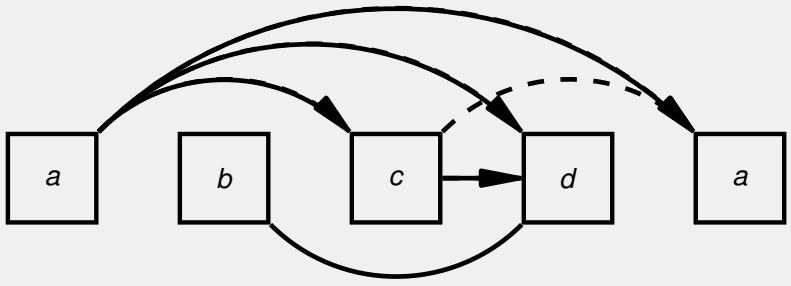


We need to complete induced **mutex edges** and **weak causality arcs**.

Saturation

Relational structure

- x = x+1; **(a)**
- y = y+3; **(b)**
- z = 2*x; **(c)**
- y = y+z; **(d)**
- x = x+1; **(a)**

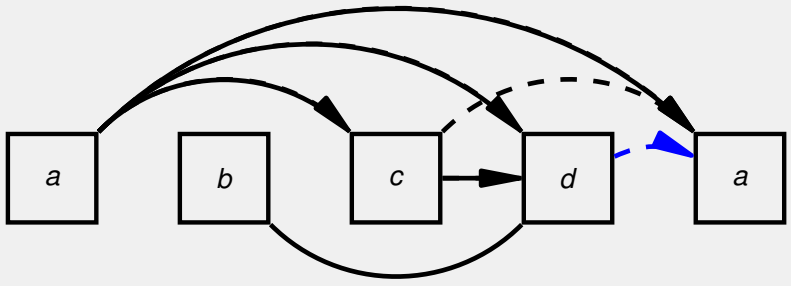


We can **specify** either new mutex edges or weak causality arcs (and **close** the structure).

Saturation

Relational structure

- x = x+1; **(a)**
- y = y+3; **(b)**
- z = 2*x; **(c)**
- y = y+z; **(d)**
- x = x+1; **(a)**

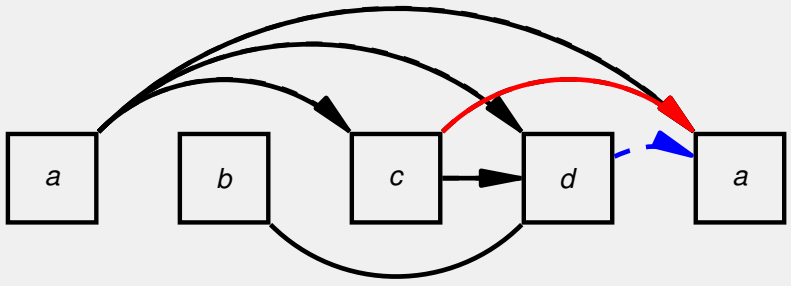


We can **specify** either new mutex edges or weak causality arcs (and **close** the structure).

Saturation

Relational structure

- x = x+1; **(a)**
- y = y+3; **(b)**
- z = 2*x; **(c)**
- y = y+z; **(d)**
- x = x+1; **(a)**

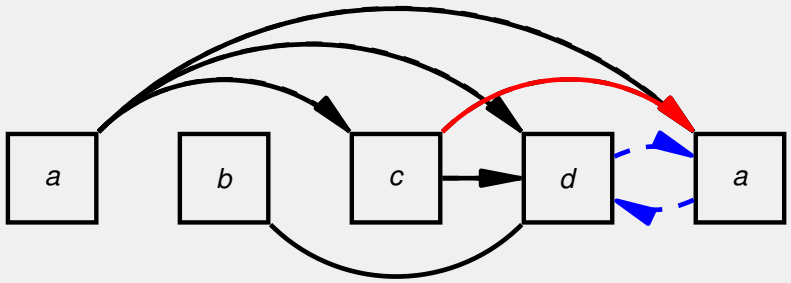


We can **specify** either new mutex edges or weak causality arcs (and **close** the structure).

Saturation

Relational structure

- x = x+1; **(a)**
- y = y+3; **(b)**
- z = 2*x; **(c)**
- y = y+z; **(d)**
- x = x+1; **(a)**

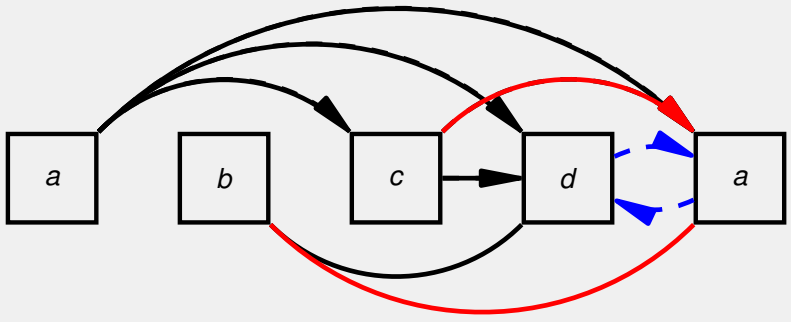


We can **specify** either new mutex edges or weak causality arcs (and **close** the structure).

Saturation

Relational structure

- x = x+1; **(a)**
- y = y+3; **(b)**
- z = 2*x; **(c)**
- y = y+z; **(d)**
- x = x+1; **(a)**

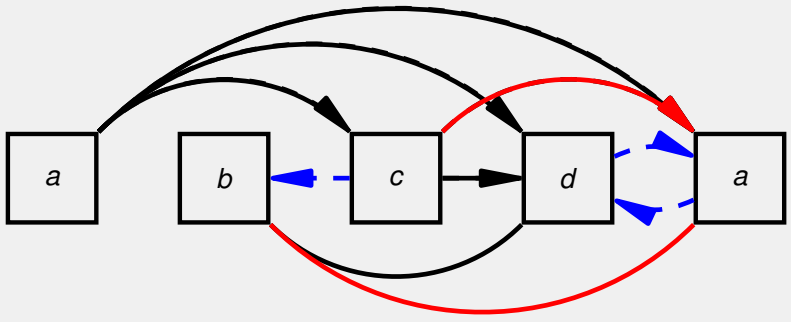


We can **specify** either new mutex edges or weak causality arcs (and **close** the structure).

Saturation

Relational structure

- x = x+1; **(a)**
- y = y+3; **(b)**
- z = 2*x; **(c)**
- y = y+z; **(d)**
- x = x+1; **(a)**

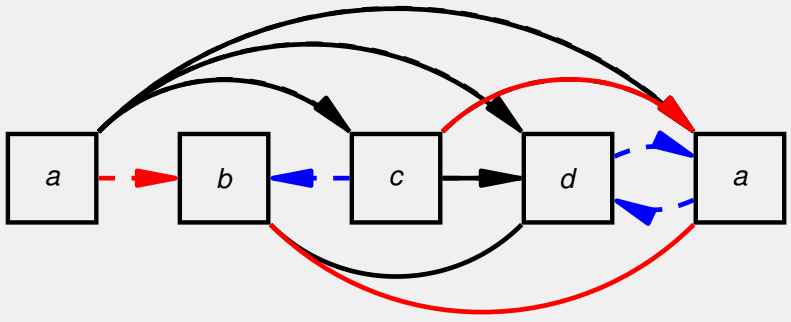


We can **specify** either new mutex edges or weak causality arcs (and **close** the structure).

Saturation

Relational structure

- x = x+1; **(a)**
- y = y+3; **(b)**
- z = 2*x; **(c)**
- y = y+z; **(d)**
- x = x+1; **(a)**

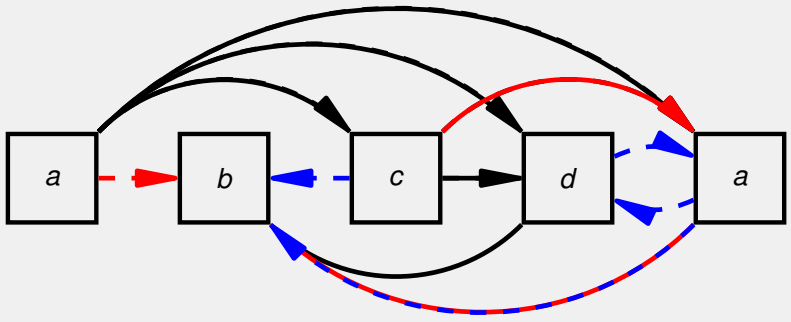


We can **specify** either new mutex edges or weak causality arcs (and **close** the structure).

Saturation

Relational structure

- x = x+1; **(a)**
- y = y+3; **(b)**
- z = 2*x; **(c)**
- y = y+z; **(d)**
- x = x+1; **(a)**

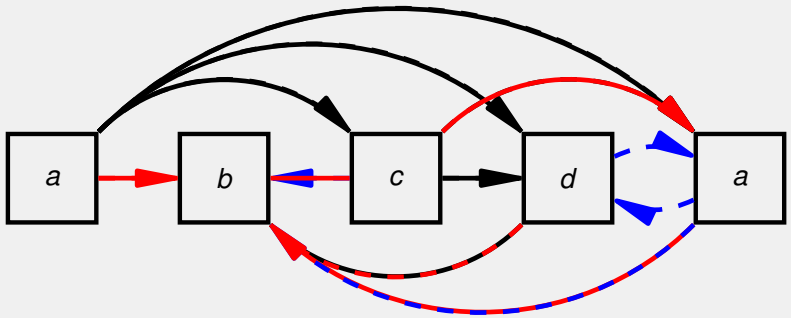


We can **specify** either new mutex edges or weak causality arcs (and **close** the structure).

Saturation

Relational structure

- x = x+1; **(a)**
- y = y+3; **(b)**
- z = 2*x; **(c)**
- y = y+z; **(d)**
- x = x+1; **(a)**

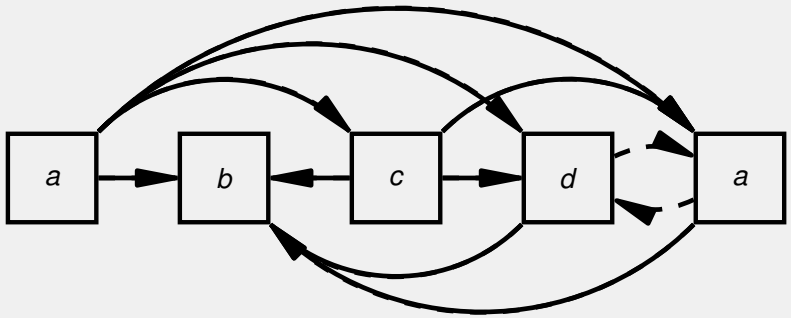


We can **specify** either new mutex edges or weak causality arcs (and **close** the structure).

Saturation

Relational structure

- x = x+1; **(a)**
- y = y+3; **(b)**
- z = 2*x; **(c)**
- y = y+z; **(d)**
- x = x+1; **(a)**



Obtained step sequence is:

ac(ad)b

Closed structures

Axioms for invariant order structures

invariant order structure is a relational structure $\langle \Delta, \rightleftharpoons, \sqsubset, \ell \rangle$ such that, for all $x, y, z, w \in \Delta$ (with $\Subset = \rightleftharpoons \cap \sqsubset$):

$$
\begin{array}{rcrcrcl}
 & & x \not\rightleftharpoons x & \wedge & x \not\sqsubset x & : & \text{IO:1} \\
 & & x \rightleftharpoons y & \Longrightarrow & y \rightleftharpoons x & : & \text{IO:2} \\
 & x \neq z \ \wedge & x \sqsubset y \sqsubset z & \Longrightarrow & x \sqsubset z & : & \text{IO:3} \\
x \Subset y \sqsubset z & \wedge & x \sqsubset y \Subset z & \Longrightarrow & x \rightleftharpoons z & : & \text{IO:4} \\
 & x \rightleftharpoons z \ \wedge & x \sqsubset y \sqsubset x & \Longrightarrow & y \rightleftharpoons z & : & \text{IO:5} \\
z \rightleftharpoons w \ \wedge & x \sqsubset w \sqsubset y & \wedge & x \sqsubset z \sqsubset y & \Longrightarrow & x \rightleftharpoons y & : \text{IO:6}
\end{array}
$$

Closure

$$\langle \Delta, \rightleftharpoons, \sqsubset, \ell \rangle \mapsto \langle \Delta, \mathsf{mut}_{\mathsf{os}}(\rightleftharpoons, \sqsubset), (\prec)^{\curlywedge}, \ell \rangle$$

- $\mathsf{mut}_{\mathsf{os}}(Q, R) = R^{\circledast} \circ Q \circ R^{\circledast} \cup R^{*} \circ_{Q} R^{*}$

Maximal structures

Axioms for layered order structures

Layered order structure is a ralational structure $\langle \Delta, \rightleftharpoons, \sqsubset, \ell \rangle$ such that, for all $x, y, z \in \Delta$:

$$
\begin{array}{lcll}
x \not\rightleftharpoons x & \wedge & x \not\sqsubset x & : \text{LO:1} \\
x \rightleftharpoons y & \Longrightarrow & y \rightleftharpoons x & : \text{LO:2} \\
x \Subset y & \Longrightarrow & y \not\sqsubset x & : \text{LO:3} \\
x \Subset y & \Longrightarrow & x \Subset z \vee z \Subset y & : \text{LO:4} \\
x \neq y & \Longrightarrow & x \sqsubset y \sqsubset x \vee x \Subset^{sym} y & : \text{LO:5}
\end{array}
$$

Equivalent objects

Layered order structures are very similar to **layered concurrent structures**, but here causality is a secondary notion.

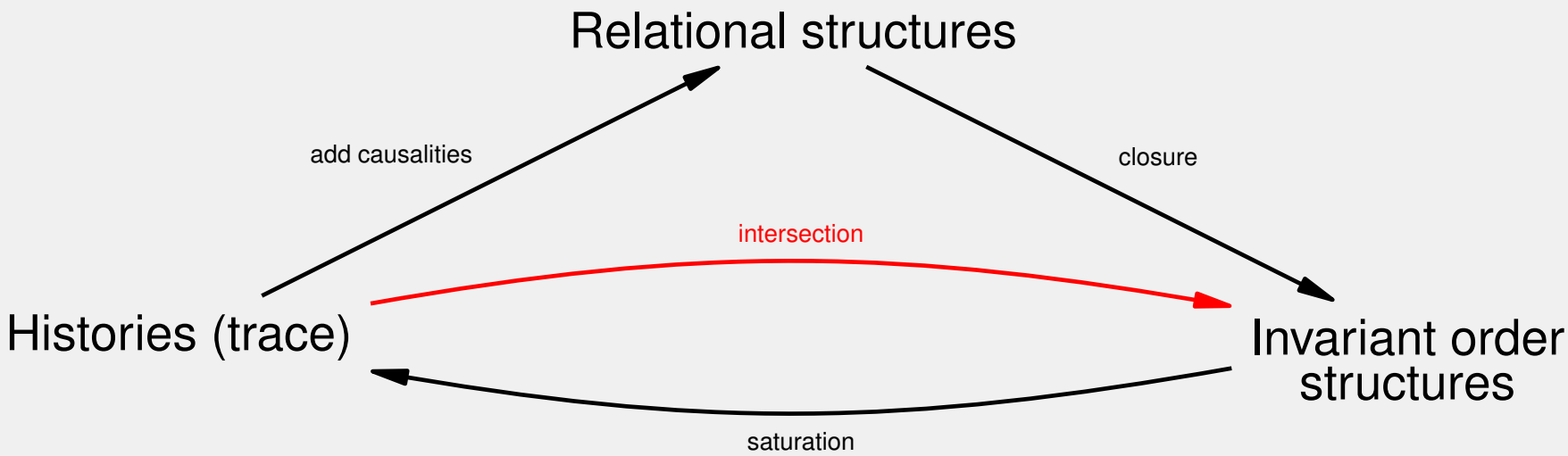
Invariant and layered order structures

Summary

- Single **observation** of a run is a step sequence of events (stratified order);
- **History** of a run is a set of possible observations;
- Order of execution together with simultaneity and sequentialisability allows to construct **dependence graph** (relational structure on events);
- **Closure** of dependence graph is an invariant order structure (order-theoretic invariant);
- **Saturation** of invariant order structure can be extended to layered order structure (any of the observations in the history);

Szpilrajn theorem

Diagram



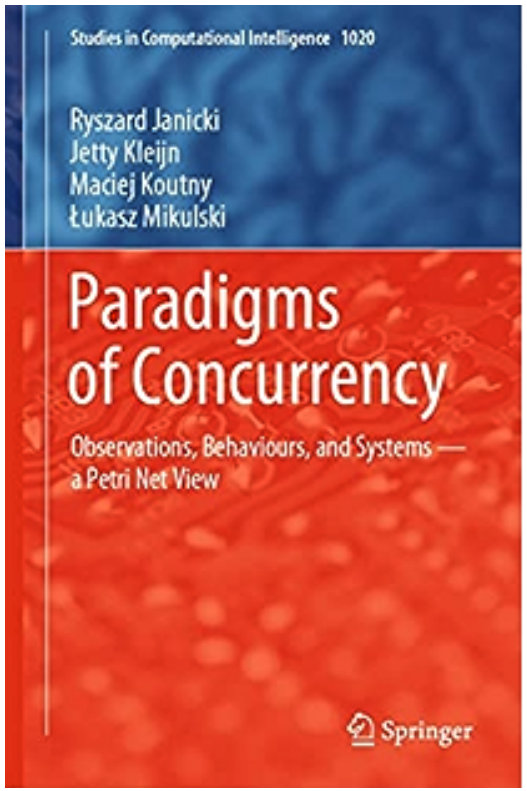
Theorem

For every invariant order structure *sos* **there exists** layered order structure *los* containing it and *sos* **is equal to the intersection** of layered order structures containing it.

What about the paradigm?

Paradigm

We drop all the requirements. We only insist that the history is closed in the set of all observations. This way we obtained **the most general** notion of invariant for step sequence semantics.



Outline

Interval orders

Norbert Wiener: A contribution to the theory of relative position [1914]

Any execution of a physical system that can be observed by a single observer must be an **interval order**.

Peter C Fishburn: Intransitive indifference with unequal indifference intervals [1970]

A countable **partial order** $(X, <)$ is interval
if and only if
there **exists a total order** $(Y, \ll)$ and two injective mappings with disjoint codomains $B, E : X \to Y$ such that forall $a, b \in X$,

- $B(a) \ll E(a)$;
- $a < b E(a) \ll B(b)$

Observations and observers

Observer (step sequences semantics)

- observes events with each event corresponding to the execution of an action;
- ~~can observe an event no more than once~~;
- may observe more than one event at the same moment;
- different executions of the same action can only be observed at different moments;
- is active for a finite time;
- eventually observes only a finite number of events;
- as long as is active, it observes all events in the system;
- the order in which events are observed by an observer respects their execution order in the system;
- reports all events it observes; and
- reports events in the order they were observed.

Observations and observers

Observer (interval order semantics)

- observes events with each event corresponding to the execution of an action;
- **can observe an event for a finite and continuous time**;
- may observe more than one event at the same moment;
- different executions of the same action can only be observed at different moments;
- is active for a finite time;
- eventually observes only a finite number of events;
- as long as is active, it observes all events in the system;
- the order in which events are observed by an observer respects their execution order in the system;
- reports all events it observes; and
- reports events in the order they were observed.

Properties of the observation

Observation is

- denoised;
- **interval**;
- concurrent;
- autoconcurrency-free;
- bounded;
- finite;
- complete;
- conform;
- unabridged; and
- reordering-free.

Leading example

Assumed execution semantics

For each action we first **read** all values and then, after a while, **write** it into a variable (lvalue).

Sample program - interval semantics

- x = x+1;
- y = y+3;
- z = 2*x;
- y = y+z;
- x = x+1;

Leading example

Assumed execution semantics

For each action we first **read** all values and then, after a while, **write** it into a variable (lvalue).

Sample program - interval semantics

- x = x+1;
- y = y+3;
- z = 2*x;
- y = y+z;
- x = x+1;

- $R_a : \{x\}$
- $R_b : \{y\}$
- $R_c : \{x\}$
- $R_d : \{y, z\}$
- $R_a : \{x\}$

- $W_a : \{x\}$
- $W_b : \{y\}$
- $W_c : \{z\}$
- $W_d : \{y\}$
- $W_a : \{x\}$

Leading example

Assumed execution semantics

For each action we first **read** all values and then, after a while, **write** it into a variable (lvalue).

Sample program - interval semantics

- x = x+1;
- y = y+3;
- z = 2*x;
- y = y+z;
- x = x+1;

- $R_a : \{x\}$
- $R_b : \{y\}$
- $R_c : \{x\}$
- $R_d : \{y, z\}$
- $R_a : \{x\}$

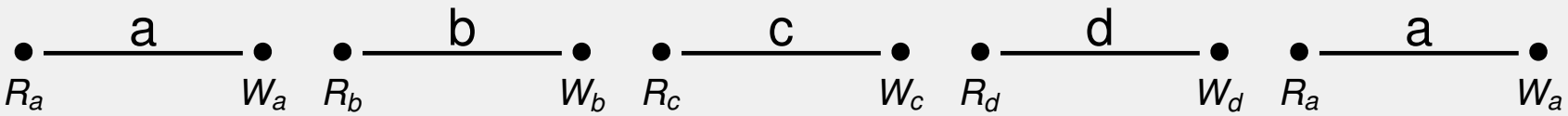
- $W_a : \{x\}$
- $W_b : \{y\}$
- $W_c : \{z\}$
- $W_d : \{y\}$
- $W_a : \{x\}$

The order of **subsequent reads** or **subsequent writes** (to different variables) does not matter.

Leading example

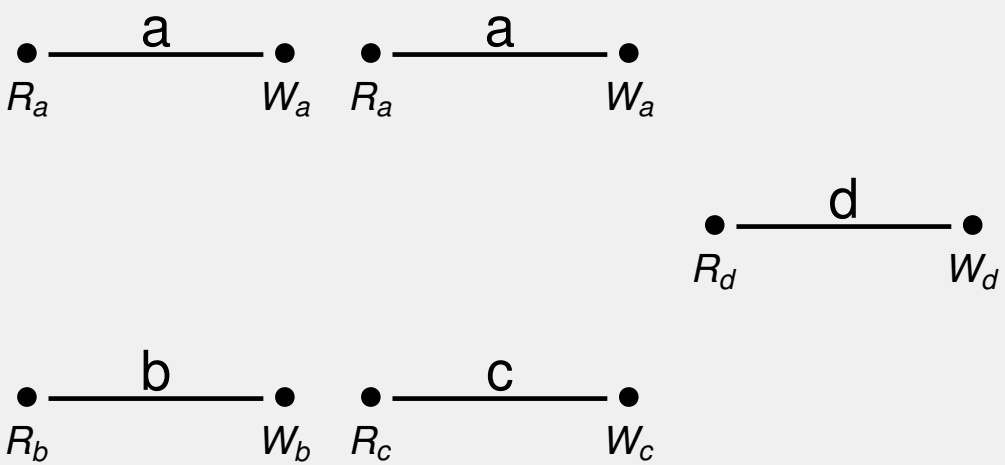
Sequence of events

- **abcda**



Step sequence of events

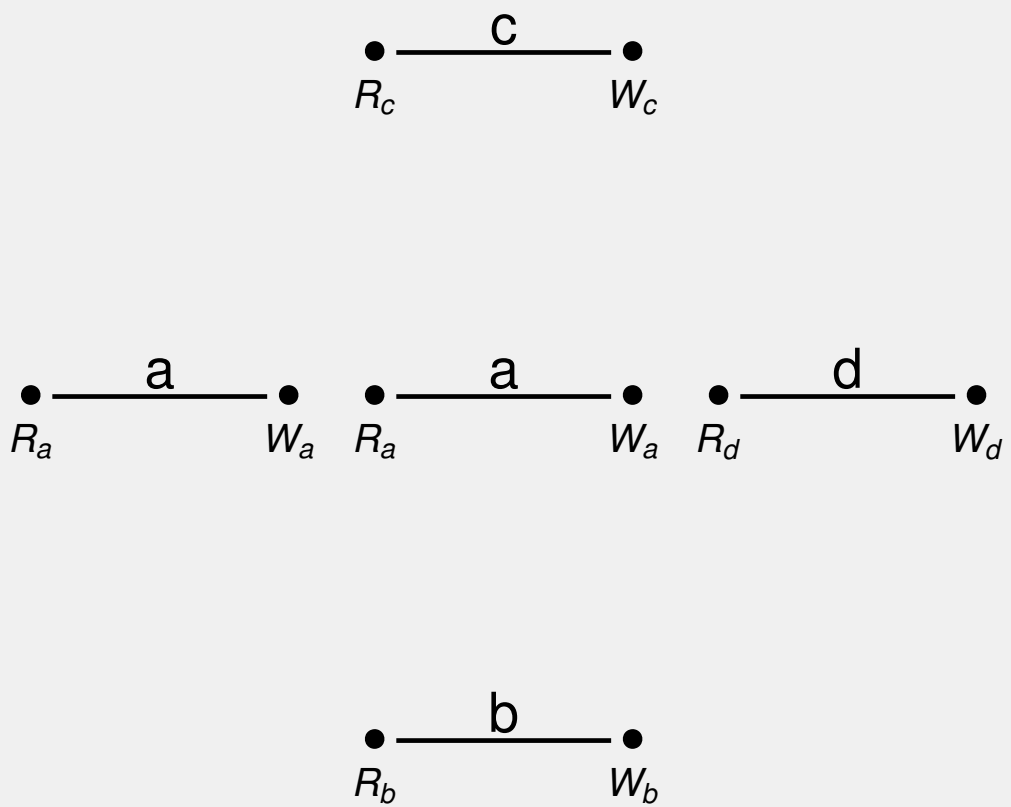
- **(ab)(ac)d**



Leading example - no mutexes

Step sequence of events

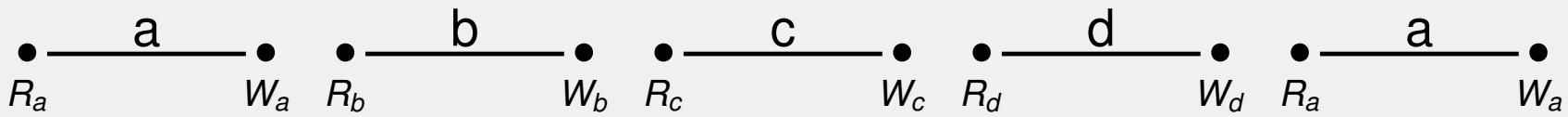
- **a(cab)d**



Leading example - total orders

Interval sequence for sequence of events

- **abcda**

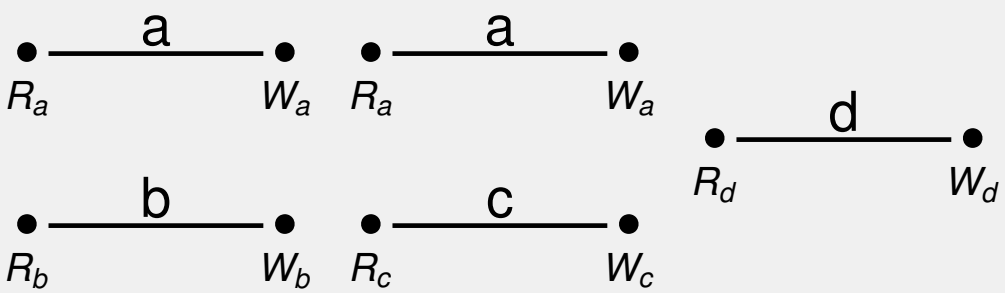


- $R_a W_a R_b W_b R_c W_c R_d W_d R_a W_a$

Leading example - stratified orders

Interval sequences for step sequence of events

- **(ab)(ac)d**

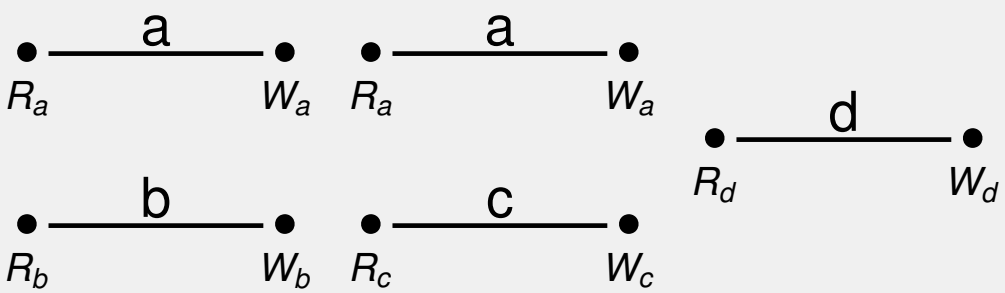


- $R_a R_b W_a W_b R_a R_c W_a W_c R_d W_d$

Leading example - stratified orders

Interval sequences for step sequence of events

- **(ab)(ac)d**

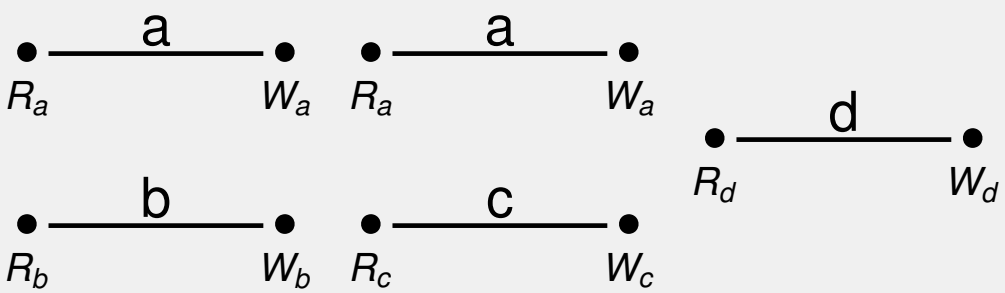


- $R_a R_b W_a W_b R_a R_c W_a W_c R_d W_d$
- $R_b R_a W_a W_b R_a R_c W_a W_c R_d W_d$

Leading example - stratified orders

Interval sequences for step sequence of events

- **(ab)(ac)d**

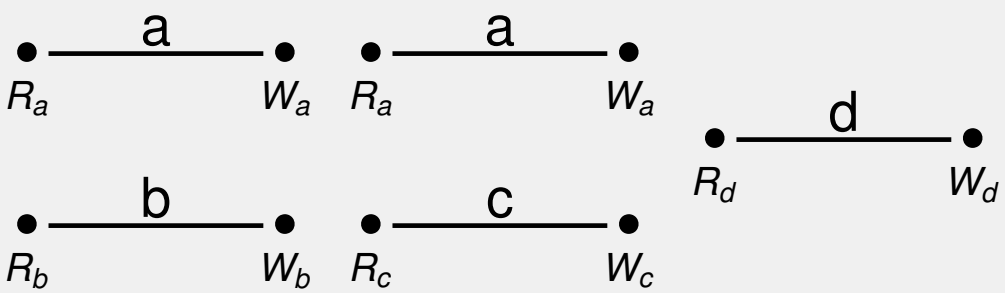


- $R_aR_bW_aW_bR_aR_cW_aW_cR_dW_d$
- $R_bR_aW_aW_bR_aR_cW_aW_cR_dW_d$
- $R_aR_bW_bW_aR_aR_cW_aW_cR_dW_d$
- $R_bR_aW_bW_aR_aR_cW_aW_cR_dW_d$

Leading example - stratified orders

Interval sequences for step sequence of events

- **(ab)(ac)d**



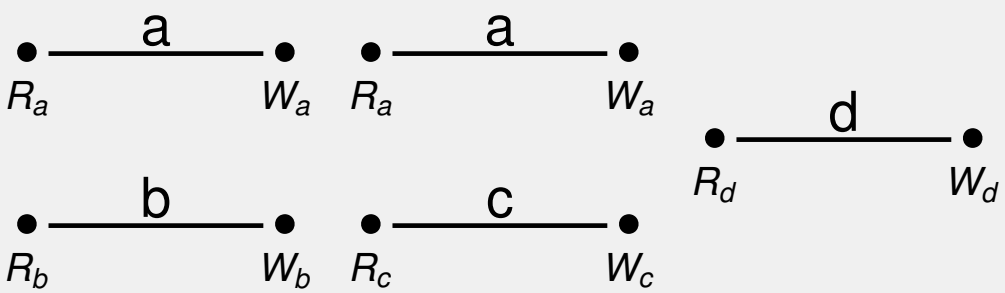
- $R_aR_bW_aW_bR_aR_cW_aW_cR_dW_d$
- $R_bR_aW_aW_bR_aR_cW_aW_cR_dW_d$
- $R_aR_bW_bW_aR_aR_cW_aW_cR_dW_d$
- $R_bR_aW_bW_aR_aR_cW_aW_cR_dW_d$
- $R_aR_bW_aW_bR_cR_aW_aW_cR_dW_d$
- $R_bR_aW_aW_bR_cR_aW_aW_cR_dW_d$
- $R_aR_bW_bW_aR_cR_aW_aW_cR_dW_d$
- $R_bR_aW_bW_aR_cR_aW_aW_cR_dW_d$

- $R_aR_bW_aW_bR_aR_cW_aW_cR_dW_d$
- $R_bR_aW_aW_bR_aR_cW_aW_cR_dW_d$
- $R_aR_bW_bW_aR_aR_cW_aW_cR_dW_d$
- $R_bR_aW_bW_aR_aR_cW_aW_cR_dW_d$
- $R_aR_bW_aW_bR_cR_aW_aW_cR_dW_d$
- $R_bR_aW_aW_bR_cR_aW_aW_cR_dW_d$
- $R_aR_bW_bW_aR_cR_aW_aW_cR_dW_d$
- $R_bR_aW_bW_aR_cR_aW_aW_cR_dW_d$

Leading example - stratified orders

Interval sequences for step sequence of events

- **(ab)(ac)d**

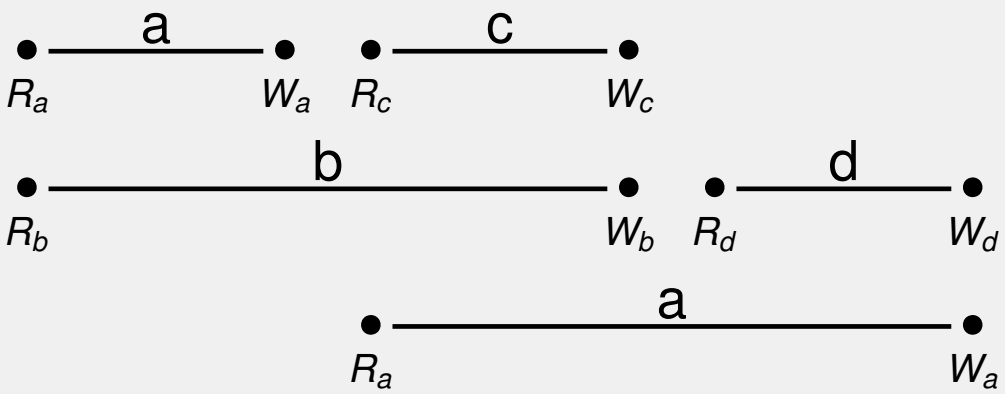


- $R_aR_bW_aW_bR_aR_cW_aW_cR_dW_d$
- $R_bR_aW_aW_bR_aR_cW_aW_cR_dW_d$
- $R_aR_bW_bW_aR_aR_cW_aW_cR_dW_d$
- $R_bR_aW_bW_aR_aR_cW_aW_cR_dW_d$
- $R_aR_bW_aW_bR_cR_aW_aW_cR_dW_d$
- $R_bR_aW_aW_bR_cR_aW_aW_cR_dW_d$
- $R_aR_bW_bW_aR_cR_aW_aW_cR_dW_d$
- $R_bR_aW_bW_aR_cR_aW_aW_cR_dW_d$

- $R_aR_bW_aW_bR_aR_cW_aW_cR_dW_d$
- $R_bR_aW_aW_bR_aR_cW_aW_cR_dW_d$
- $R_aR_bW_bW_aR_aR_cW_aW_cR_dW_d$
- $R_bR_aW_bW_aR_aR_cW_aW_cR_dW_d$
- $R_aR_bW_aW_bR_cR_aW_aW_cR_dW_d$
- $R_bR_aW_aW_bR_cR_aW_aW_cR_dW_d$
- $R_aR_bW_bW_aR_cR_aW_aW_cR_dW_d$
- $R_bR_aW_bW_aR_cR_aW_aW_cR_dW_d$

Leading example - interval orders

Interval sequences for interval order (not stratified order)



- $R_aR_bW_aR_aR_cW_bW_cR_dW_aW_d$
- $R_bR_aW_aR_aR_cW_bW_cR_dW_aW_d$
- $R_aR_bW_aR_cR_aW_bW_cR_dW_aW_d$
- $R_bR_aW_aR_cR_aW_bW_cR_dW_aW_d$
- $R_aR_bW_aR_aR_cW_cW_bR_dW_aW_d$
- $R_bR_aW_aR_aR_cW_cW_bR_dW_aW_d$
- $R_aR_bW_aR_cR_aW_cW_bR_dW_aW_d$
- $R_bR_aW_aR_cR_aW_cW_bR_dW_aW_d$
- $R_aR_bW_aR_aR_cW_bW_cR_dW_dW_a$
- $R_bR_aW_aR_aR_cW_bW_cR_dW_dW_a$
- $R_aR_bW_aR_cR_aW_bW_cR_dW_dW_a$
- $R_bR_aW_aR_cR_aW_bW_cR_dW_dW_a$
- $R_aR_bW_aR_aR_cW_cW_bR_dW_dW_a$
- $R_bR_aW_aR_aR_cW_cW_bR_dW_dW_a$
- $R_aR_bW_aR_cR_aW_cW_bR_dW_dW_a$
- $R_bR_aW_aR_cR_aW_cW_bR_dW_dW_a$

Mazurkiewicz traces over beginnings and ends

Formal definition

- Alphabet of actions: $\Sigma_{\Downarrow\Uparrow} = \{R_a, W_a, R_b, W_b, R_c, W_c, R_d, W_d\}$;
- Interval orders independence: $\mathrm{ind}_{int} = \{\langle R_a, R_b\rangle, \langle W_a, W_b\rangle \mid a \neq b\}$;
- Concurrent alphabets: $\Phi_{int} = \langle \Sigma_{\Downarrow\Uparrow}, \mathrm{ind}_{int}\rangle$;
- Immediate similarity: $\approx_{\Phi_{int}} \subseteq \Sigma^*_{\Downarrow\Uparrow} \times \Sigma^*_{\Downarrow\Uparrow}$;
- Equivalence relation: $\equiv_{\Phi_{int}} \subseteq \Sigma^*_{\Downarrow\Uparrow} \times \Sigma^*_{\Downarrow\Uparrow} = \approx^*_{\Phi_{int}}$
- Mazurkiewicz traces (equivalence classes): $\Sigma^*_{\Downarrow\Uparrow} / \equiv_{\Phi_{int}}$;
- Mazurkiewicz traces over $\Sigma_{\Downarrow\Uparrow}$ are equivalent to interval orders over $\Sigma = \{a, b, c, d\}$, notation $\wp$;

Leading example - without mutexes

Equivalent executions - total and stratified orders

- abcad
- abcda
- acabd
- acbad
- acbda
- bacad
- bacda
- (ab)cad
- abc(ad)
- ac(ab)d
- acb(ad)
- a(cb)da
- bac(ad)
- (ab)cda
- (ab)c(ad)
- a(bc)(ad)
- ab(ca)d
- a(ca)bd
- ba(ca)d
- (ab)(ca)d
- a(cab)d

Leading example - no mutexes

Dependencies on beginnings and ends

We cannot move **reads** of some variables before their recent **writes** (and need to distinguish between two executions).

Leading example

$$R_a : \{x\} \text{ ———— } W_a : \{x\}$$

$$R_b : \{y\} \text{ ———— } W_b : \{y\}$$

$$R_c : \{x\} \text{ ———— } W_c : \{z\}$$

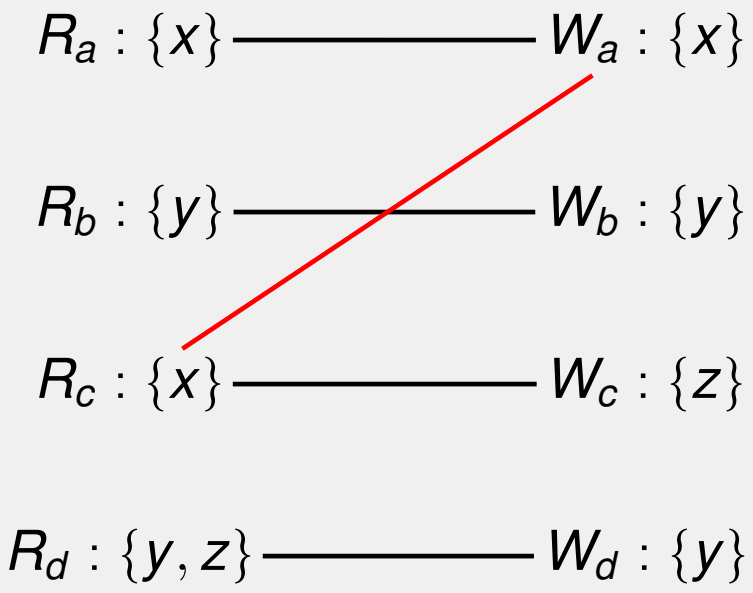
$$R_d : \{y, z\} \text{ ———— } W_d : \{y\}$$

Leading example - no mutexes

Dependencies on beginnings and ends

We cannot move **reads** of some variables before their recent **writes** (and need to distinguish between two executions).

Leading example

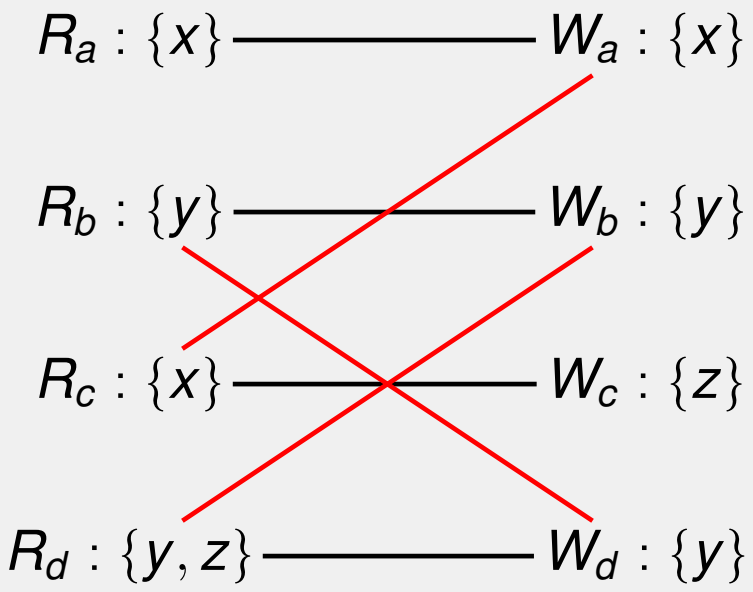


Leading example - no mutexes

Dependencies on beginnings and ends

We cannot move **reads** of some variables before their recent **writes** (and need to distinguish between two executions).

Leading example

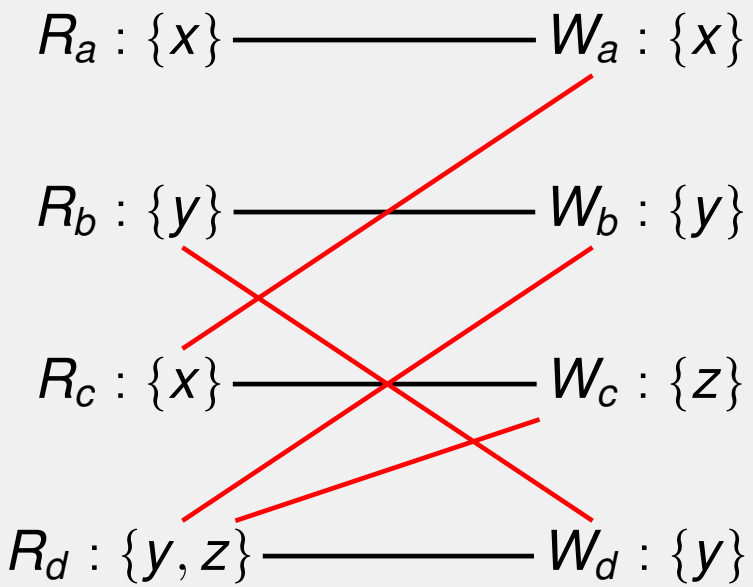


Leading example - no mutexes

Dependencies on beginnings and ends

We cannot move **reads** of some variables before their recent **writes** (and need to distinguish between two executions).

Leading example



Interval traces

Formal definition

- Alphabet of actions: $\Sigma = \{a, b, c, d\}$;
- Weak independence relation:
  $\text{wind} = \{\langle a, b\rangle^2, \langle c, a\rangle, \langle a, d\rangle^2, \langle b, c\rangle^2, \langle d, c\rangle\}$;
- Independence relation: $\text{ind}_{\Downarrow\Uparrow} = \text{ind}_{int} \cup$
  $\{\langle R_a, W_b\rangle^2, \langle R_a, W_c\rangle^2, \langle R_a, W_d\rangle^2, \langle R_b, W_a\rangle^2,$
  $\langle R_b, W_c\rangle^2, \langle R_c, W_b\rangle^2, \langle R_c, W_d\rangle^2, \langle R_d, W_a\rangle^2\}$;
- Concurrent alphabets: $\Phi_{int} = \langle \Sigma_{\Downarrow\Uparrow}, \text{ind}_{int}\rangle$; $\Phi_{\Downarrow\Uparrow} = \langle \Sigma_{\Downarrow\Uparrow}, \text{ind}_{\Downarrow\Uparrow}\rangle$;
- Interval orders: $\wp = {}^{\Sigma^*_{\Downarrow\Uparrow}}/_{\equiv_{\Phi_{int}}}$;
- Traces (considering weak independence): ${}^{\Sigma^*_{\Downarrow\Uparrow}}/_{\equiv_{\Phi_{\Downarrow\Uparrow}}}$;
- Equivalence relation $\equiv_{\Phi_{\Downarrow\Uparrow}}$ is an invariant for $\wp$ ($\equiv_{\Phi_{int}} \subseteq \equiv_{\Phi_{\Downarrow\Uparrow}}$);
- Interval traces: ${}^{\wp}/_{\equiv_{\Phi_{\Downarrow\Uparrow}}}$.

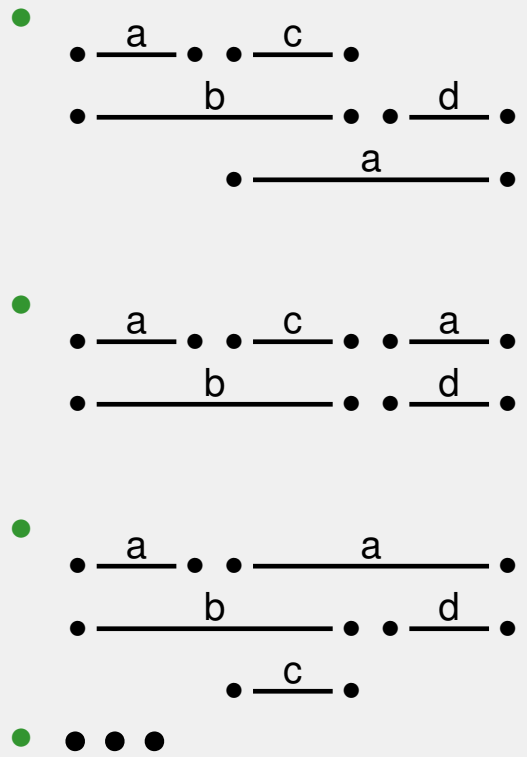
Leading example - no mutexes

Equivalent executions

- abcad
- abcda
- acabd
- acbad
- acbda
- bacad
- bacda

- (ab)cad
- abc(ad)
- ac(ab)d
- acb(ad)
- a(cb)da
- bac(ad)
- (ab)cda
- (ab)c(ad)
- a(bc)(ad)

- ab(ca)d
- a(ca)bd
- ba(ca)d
- (ab)(ca)d
- a(cab)d



Closed structures

Axioms for invariant combined structures

invariant combined structure is a tuple $\langle \Delta, \prec, \sqsubset, \ell \rangle$ such that, for all $x, y, z, w \in \Delta$

$$
\begin{array}{lrcll}
 & x \not\sqsubset x & & & : \text{IC:1} \\
 & x \prec y & \Longrightarrow & x \sqsubset y \not\sqsubset x & : \text{IC:2} \\
 & x \prec y \prec z & \Longrightarrow & x \prec z & : \text{IC:3} \\
x \sqsubset y \prec z \quad \vee & x \prec y \sqsubset z & \Longrightarrow & x \sqsubset z & : \text{IC:4} \\
 & x \prec y \sqsubset z \prec w & \Longrightarrow & x \prec w & : \text{IC:5} \\
 & x \sqsubset y \prec z \sqsubset w \neq x & \Longrightarrow & x \sqsubset w & : \text{IC:6}
\end{array}
$$

Maximal structures

Axioms for interval poset structures

interval order structure is a tuple $\langle \Delta, \prec, \sqsubset, \ell \rangle$ such that, for all $x, y, z, w \in \Delta$

$$
\begin{array}{rcl}
x \not\sqsubset x & & : \ \text{IP:1} \\
x \prec y & \Longrightarrow \quad x \sqsubset y & : \ \text{IP:2} \\
x \prec y & \Longleftrightarrow \quad y \not\sqsubset x \neq y & : \ \text{IP:3} \\
x \prec y \ \wedge \ z \prec w & \Longrightarrow \quad x \prec w \ \vee \ z \prec y & : \ \text{IP:4}
\end{array}
$$

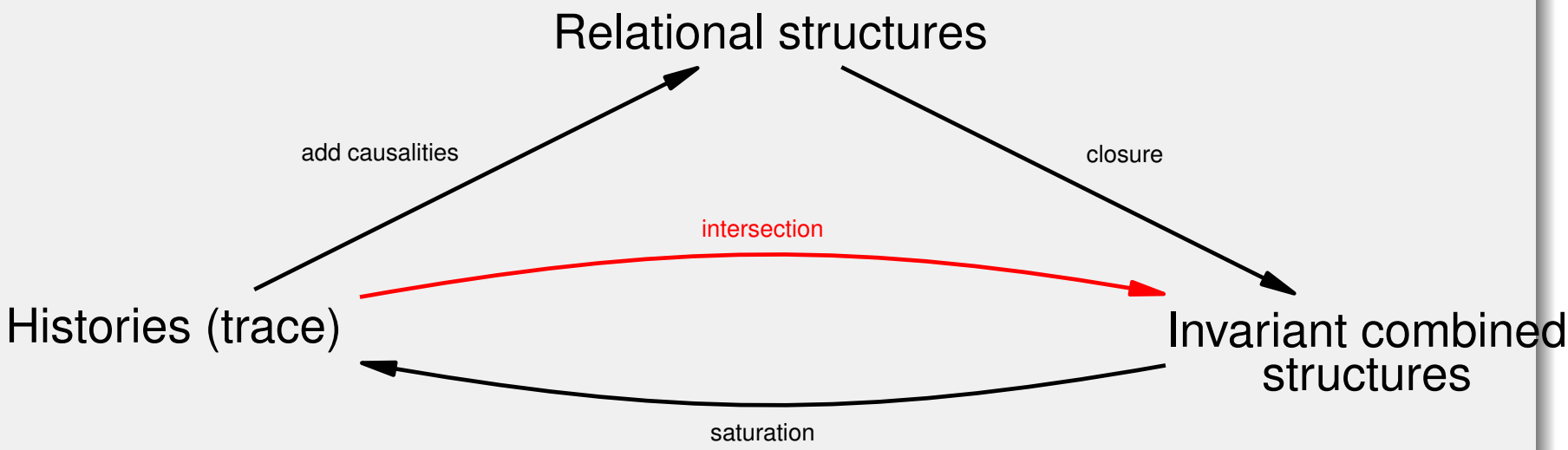
Invariant and layered order structures

Summary

- Single **observation** of a run is a interval partial order of events;
- **History** of a run is a set of possible observations;
- Order of execution together with weak independence allows to construct **dependence graph** (relational structure on events);
- **Closure** of dependence graph is an invariant combined structure;
- **Saturation** of invariant order structure can be extended to interval poset structure (any of the observations in the history);

Szpilrajn theorem

Diagram



Theorem

For every invariant combined structure *ios* **there exists** interval poset structure *ips* containing it and *ios* **is equal to the intersection** of interval poset structures containing it.

Further generalisation

Question

What about mutexes?

Further generalisation

Question

What about mutexes?

Work in progress

- Ryszard Janicki, Maciej Koutny, Łukasz Mikulski:
  Interval Traces with Mutex Relation. Petri Nets 2023: 145-166

Summary

Semantics

- sequential;
- step sequence;
- interval order.

Forbidden cycles

- acyclic graphs (no cycles of $\prec$);
- separable structures (no cycle of $\sqsubset$ is connected by $\rightleftharpoons$);
- combined interval acyclic structures
  (every cycle has two consecutive $\sqsubset$ arcs).

Summary

Petri net systems

- elementary systems
- elementary systems with activators (step sequence semantics)
- elementary systems with activators and mutexes
- elementary systems with activators (interval order semantics)

Traces

- Mazurkiewicz traces
- Combined traces
- Step traces
- Interval traces

Summary

Milestones

- we started from Mazurkiewicz concept of equivalent computations and partial orders as invariants [1977];
- adapting true concurrency paradigm allows us to elevate to step sematnics [1987];
- weakening causality made it possible to introduce the notion of combined traces [1993];
- decomposition of causality to weak causality and mutual exclusion led to the definition of step traces [2013]
- rejection of the transitivity of simultaneity allows us to elevate to interval semantics [2009-2023]
- what next?

Summary

Design clues

- Acyclicity notion (allowed structures) are crucial design decision;
- Saturation is a secondary concept based on acyclity;
- Closed structure and dependence structures are secondary concepts based on saturation;

Future work

- More precise notion of acyclicity for interval semantics;
- Mutexes for interval semantics;
- Net systems for interval semantics;