University of Cyprus

Reversible Computation in Truly Concurrent Models of Distributed Systems

Anna Philippou

Joint work with Kyriaki Psara

Department of Computer Science, University of Cyprus

18 March 2026, Università degli Studi di Urbino Carlo Bo

Talk Synopsis

1. Introduction
   - Petri Nets
   - Reversibility
2. Reversing Petri nets
   - RPNs and the Three Types of Reversibility
   - Multi-Token RPNs under the Individual-Token Interpretation
   - Multi-token RPNs under the Collective-Token Interpretation
3. Conclusions

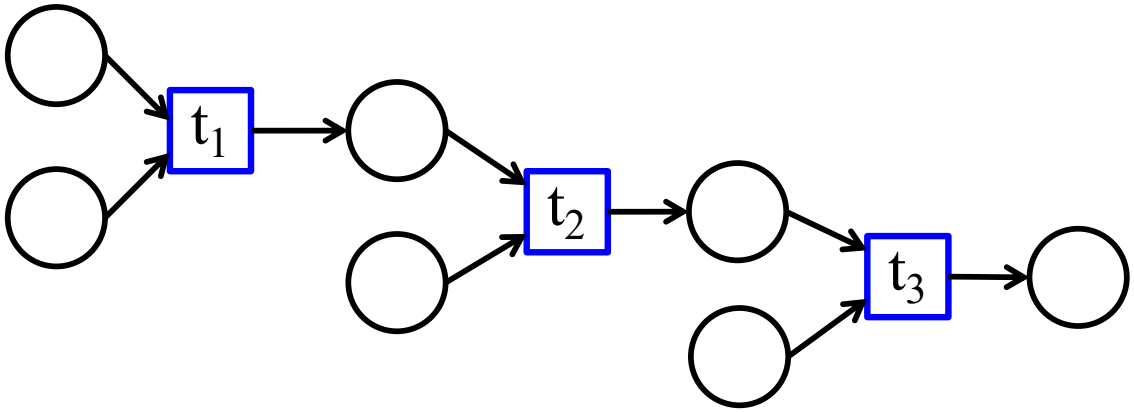
Introduction – Petri nets

Petri nets

- A powerful graphical language for discrete event systems [Petri 1962]
- Rich mathematical theory
- Suitable for modeling: concurrency, distribution, synchronization, precedence and priority, asynchronous behavior, timed and/or stochastic systems

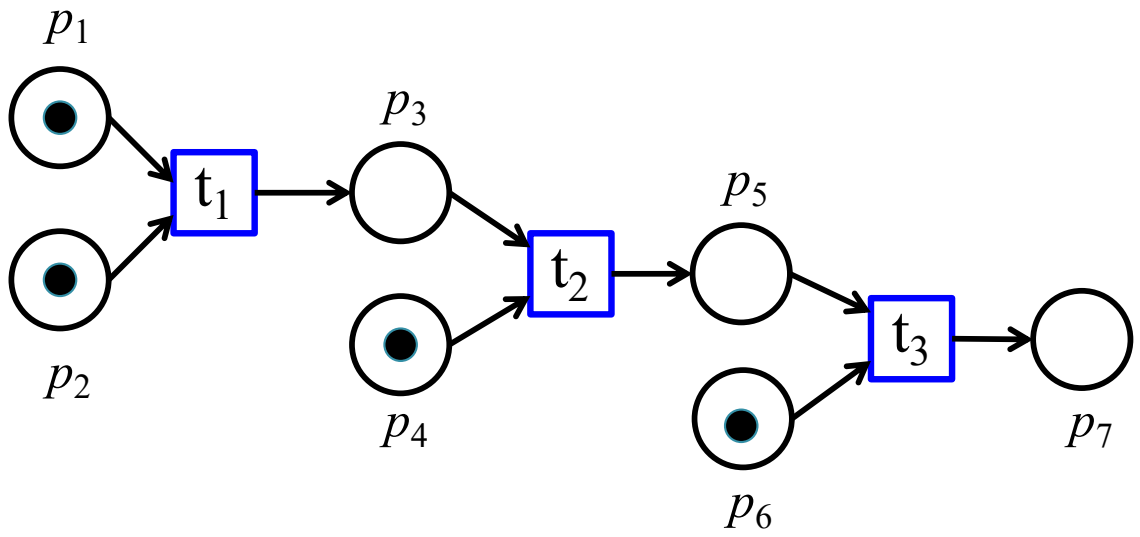
Informal Definition

- The graphical representation of a PN is a bipartite graph
- Two kinds of nodes:
  - Places: usually model resources or partial state of a system
  - Transitions: model state change and synchronization
- Arcs: connect nodes of different types



Petri net state

- The state of a system is modelled by marking the places with tokens
- A place can be marked with a finite (possibly zero) number of tokens



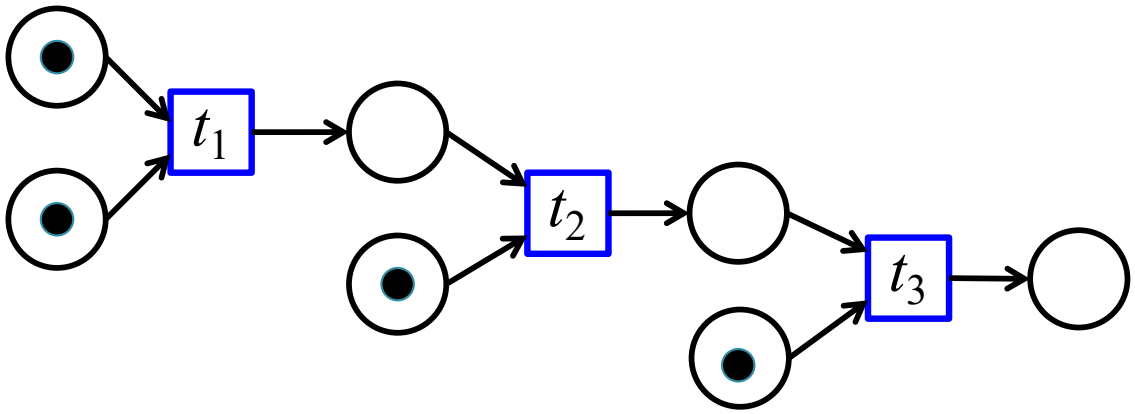
- *Marking, M*: A function that assigns tokens to places

$M(p_1) = 1$ $M(p_2) = 1$ $M(p_3) = 0$ $M(p_4) = 1$

$M(p_5) = 0$ $M(p_6) = 1$ $M(p_7) = 0$

Petri net execution

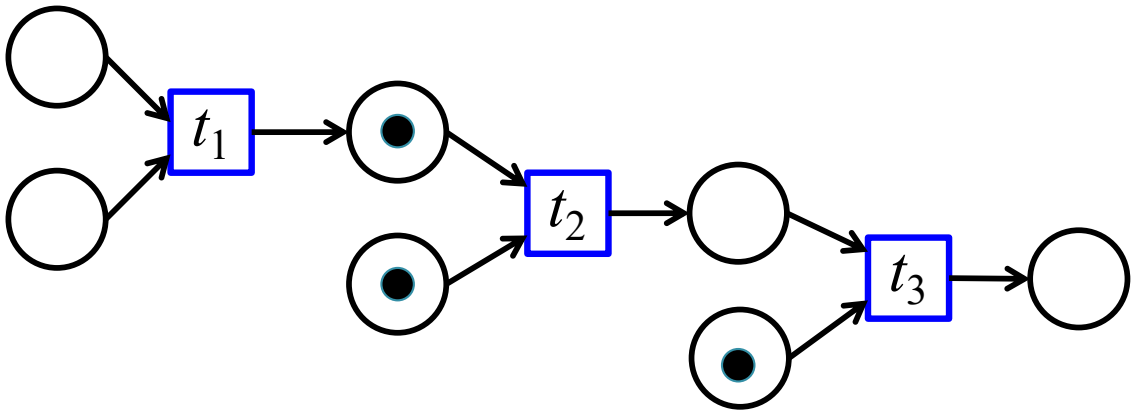
- A transition ***t*** is *enabled* in a certain marking if
  - Every incoming place of ***t*** contains at least one token in the marking
- An enabled transition can *fire* and results in a new marking
- The effect of firing a transition ***t*** is:
  - to subtract a token from each of the incoming places of ***t***, and
  - to add a token to each of the outgoing places of ***t***



(1,1,0,1,0,1,0)

Petri net execution

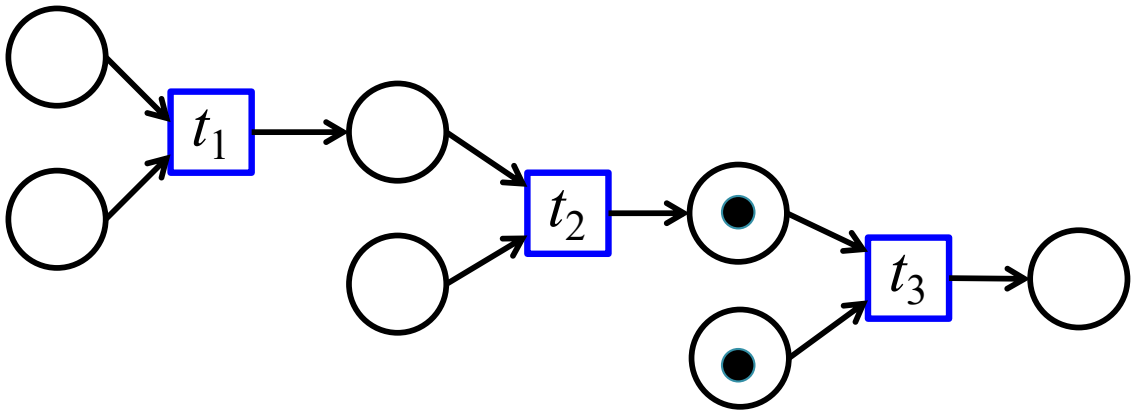
- A transition ***t*** is *enabled* in a certain marking if
  - Every incoming place of ***t*** contains at least one token in the marking
- An enabled transition can *fire* and results in a new marking
- The effect of firing a transition ***t*** is:
  - to subtract a token from each of the incoming places of ***t***, and
  - to add a token to each of the outgoing places of ***t***



$(1,1,0,1,0,1,0) \rightarrow (0,0,1,1,0,1,0)$

Petri net execution

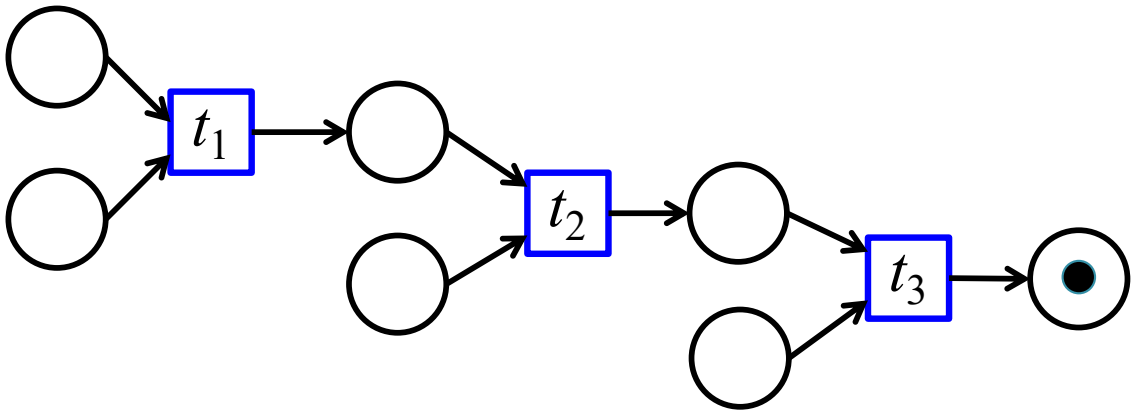
- A transition ***t*** is *enabled* in a certain marking if
  - Every incoming place of ***t*** contains at least one token in the marking
- An enabled transition can *fire* and results in a new marking
- The effect of firing a transition ***t*** is:
  - to subtract a token from each of the incoming places of ***t***, and
  - to add a token to each of the outgoing places of ***t***



$(1,1,0,1,0,1,0) \rightarrow (0,0,1,1,0,1,0) \rightarrow (0,0,0,0,1,1,0)$

Petri net execution

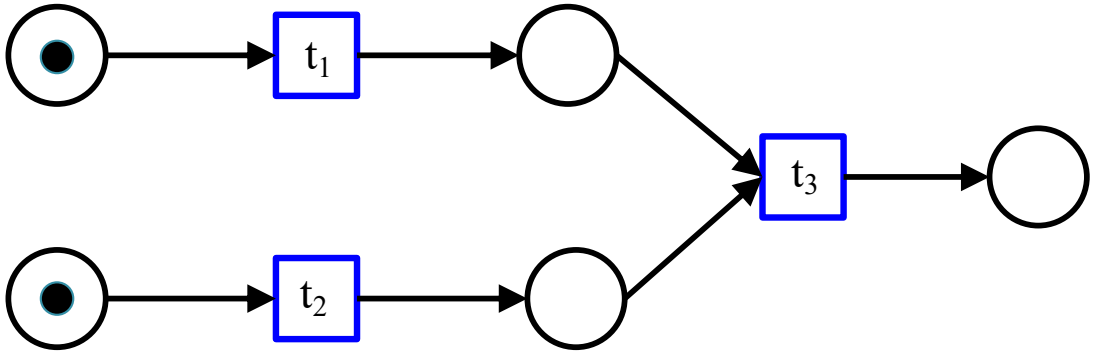
- A transition ***t*** is *enabled* in a certain marking if
  - Every incoming place of ***t*** contains at least one token in the marking
- An enabled transition can *fire* and results in a new marking
- The effect of firing a transition ***t*** is:
  - to subtract a token from each of the incoming places of ***t***, and
  - to add a token to each of the outgoing places of ***t***



$(1,1,0,1,0,1,0) \rightarrow (0,0,1,1,0,1,0) \rightarrow (0,0,0,0,1,1,0) \rightarrow (0,0,0,0,0,0,1)$

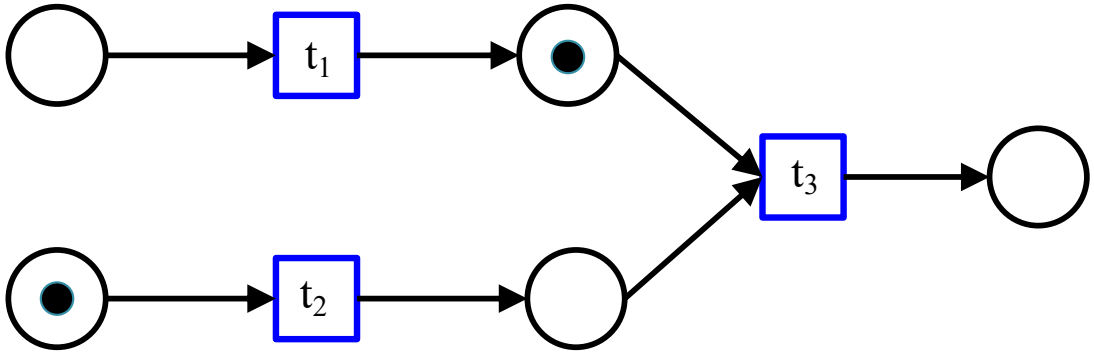
Causality and Concurrency

- Petri nets distinguish causality and independence
- Causality captures the dependency between events: one transition must fire before another can
- Conflict occurs when transitions compete for tokens
- Concurrency describes transitions that can fire simultaneously because they share no causal or conflict relationship



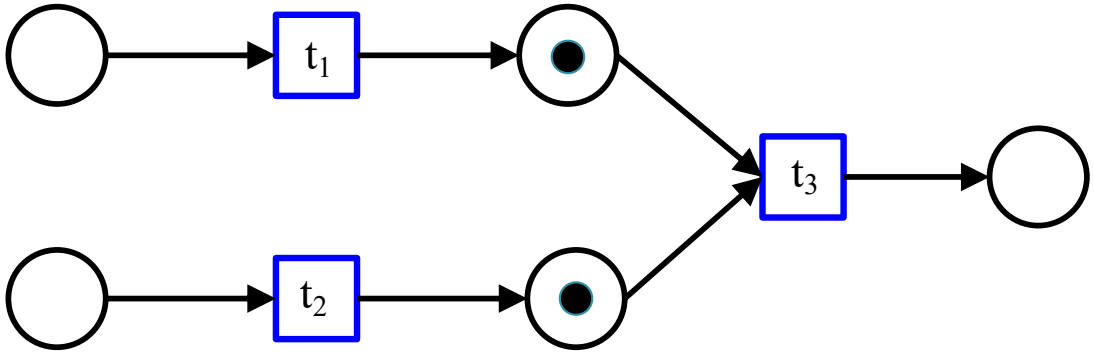
Causality and Concurrency

- Petri nets distinguish causality and independence
- Causality captures the dependency between events: one transition must fire before another can.
- Conflict occurs when transitions compete for tokens
- Concurrency describes transitions that can fire simultaneously because they share no causal or conflict relationship



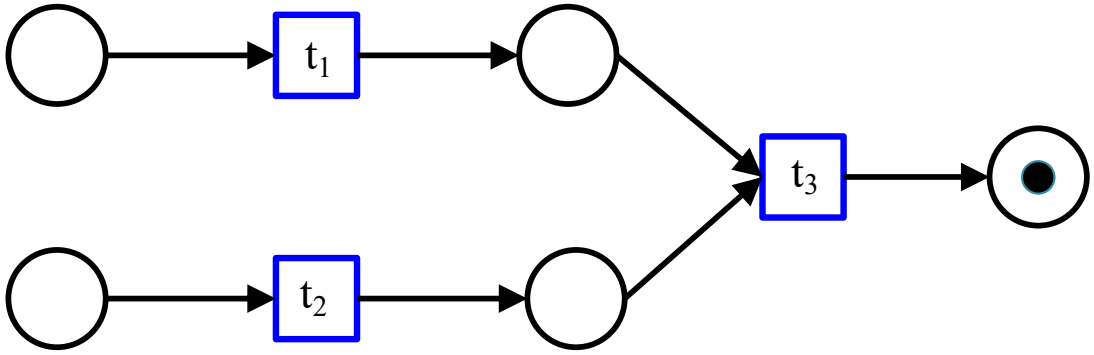
Causality and Concurrency

- Petri nets distinguish causality and independence
- Causality captures the dependency between events: one transition must fire before another can.
- Conflict occurs when transitions compete for tokens
- Concurrency describes transitions that can fire simultaneously because they share no causal or conflict relationship



Causality and Concurrency

- Petri nets distinguish causality and independence
- Causality captures the dependency between events: one transition must fire before another can.
- Conflict occurs when transitions compete for tokens
- Concurrency describes transitions that can fire simultaneously because they share no causal or conflict relationship



Applications areas

- Distributed algorithms and protocols
- Workflow management and business processes
- Asynchronous circuit verification
- Biological pathway and chemical reaction modelling
- Foundation for extensions such as timed, coloured, and stochastic Petri nets

Analysis Methods

- Reachability graph exploration
- Invariant analysis (place and transition invariants)
- Deadlock and liveness checking
- Model checking of temporal properties

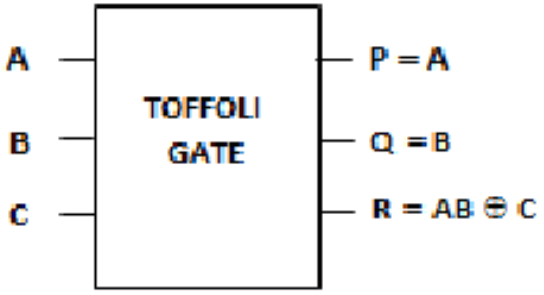
Introduction – Reversible computation

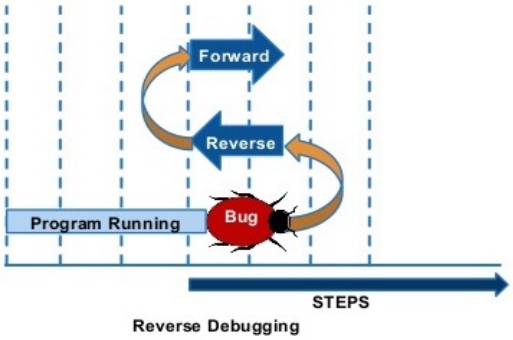
Reversible computation

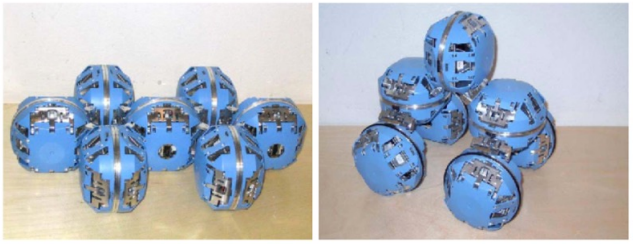
- Unconventional form of computing where computation can run backwards as naturally as it can go forwards
  - Every set of instructions can be carried out in reverse order
  - At any point of execution, it is possible to identify both the next and the previous state
- Origins
  - Landauer, 1960: logically irreversible operations result in bit erasure that causes heat dissipation, i.e., loss of energy
    - Whenever a computer throws away bits of information it generates entropy
    - It dissipates at least $kT\ln 2$ of energy for each bit of information it erases
  - Reversible logic gates and circuits may lead to low-energy computing

Motivation

- The ever-increasing demand for energy
  - design of reversible logic gates and circuits leading to low-power computing
- Program debugging and testing
  - Bring a program to a state where certain conditions are met
- Programming abstractions for fault-tolerant systems
  - Transactions, system-recovery schemes, checkpoint-rollback protocols
- Applications featuring reversible behavior
  - Biological modeling: biochemical reactions are inherently reversible
  - Robotics, etc.







(a) Cluster (b) Car

Formal models for reversible computation

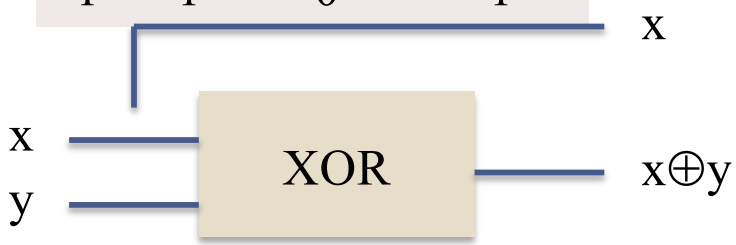
- Investigate the theoretical foundations of reversibility
- Develop models for reversible systems and techniques for their analysis
- Aid towards understanding, modeling, and implementing reversible actions as a feature of computation
- Reversible Models of Computation:
  - Turing Machines [Lecerf 1963, Morita&Yamaguchi 2007], Automata [Pin 1992], etc.
  - Models of concurrency:
    - process calculi [Danos&Krivine 2004, Philips&Ulidowski 2016, Lanese,Mezzina,Stefani 2010, Bernardo,Mezzina,Esposito, 2026, Esposito,Aldini,Bernardo,Rossi, 2025], etc.
    - event structures [Philips,Ulidowski,Yuen 2014, Melgratti,Mezzina,Pinna 2024], etc.
    - Petri nets [Melgratti, Mezzina, Ulidowski 2020, Barylska,Koutny,Mikulski,Piatkowski, 2016], etc.
  - Programming Languages: Janus [Yokoyama & Gluck 2007], r-fun [Yokoyama et al. 2011], etc.

Need for “memories”

- Two-way determinism – one-to-one mapping between inputs and outputs
  - the operator NOT is reversible
- The operator XOR is *not* reversible

input		output
x	y	x ⊕ y
0	0	0
0	1	1
1	0	1
1	1	0

input		output	
x	y	x ⊕ y	x
0	0	0	0
0	1	1	0
1	0	1	1
1	1	0	1



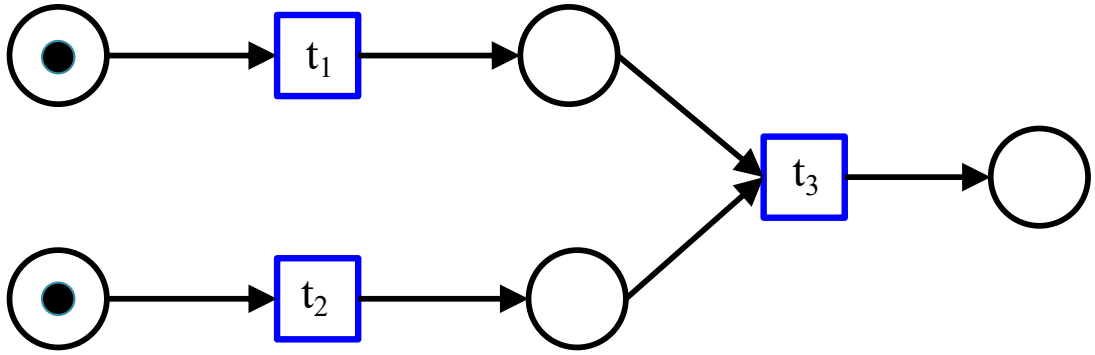
Reversibility types

- *Backtracking*: Rewinding the computation trace in the exact reverse order
  - Too restrictive in the context of concurrent computation
  - Induces fake causal dependencies
- *Causal-order reversibility*: An event can reverse only if all its effects have been reversed beforehand
  - More flexible than backtracking
- *Out-of-causal-order reversibility*: Allows undoing actions in an out-of-causal order
  - Inherent in systems such as biochemical reactions
  - Desirable in applications such as system reliability

Reversing Petri nets - RPNs and the Three Types of Reversibility

Reversibility types – Backtracking

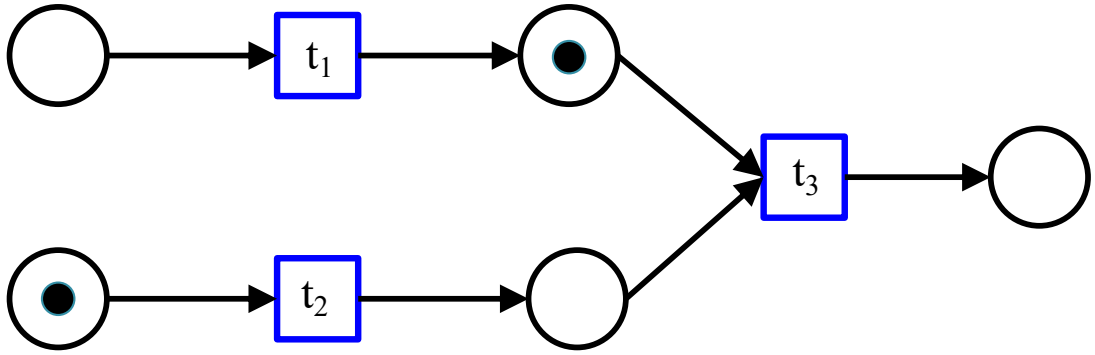
- Rewinding the computation trace
  - Exact reverse order
- Example



Forward execution: $\langle t_1, t_2, t_3 \rangle$

Reversibility types – Backtracking

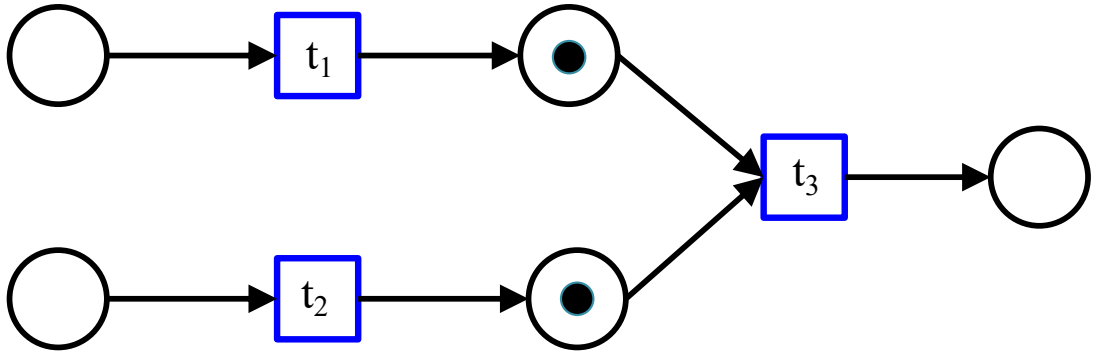
- Rewinding the computation trace
  - Exact reverse order
- Example



Forward execution: $\langle t_1, t_2, t_3 \rangle$

Reversibility types – Backtracking

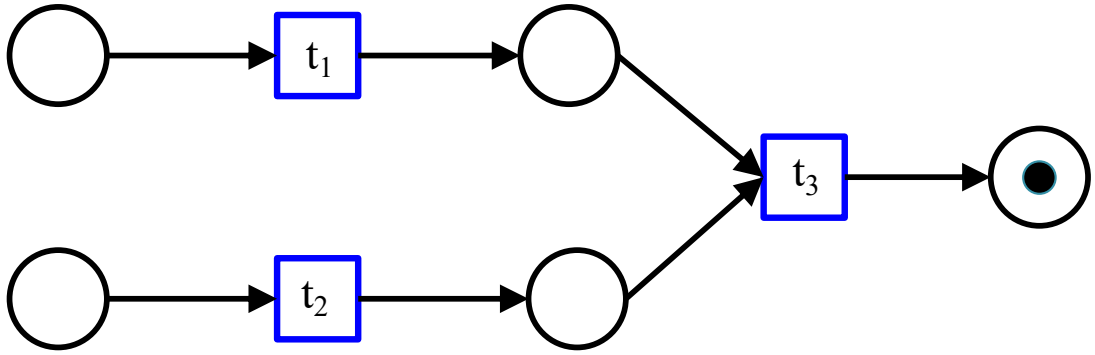
- Rewinding the computation trace
  - Exact reverse order
- Example



Forward execution: $\langle t_1, t_2, t_3 \rangle$

Reversibility types – Backtracking

- Rewinding the computation trace
  - Exact reverse order
- Example

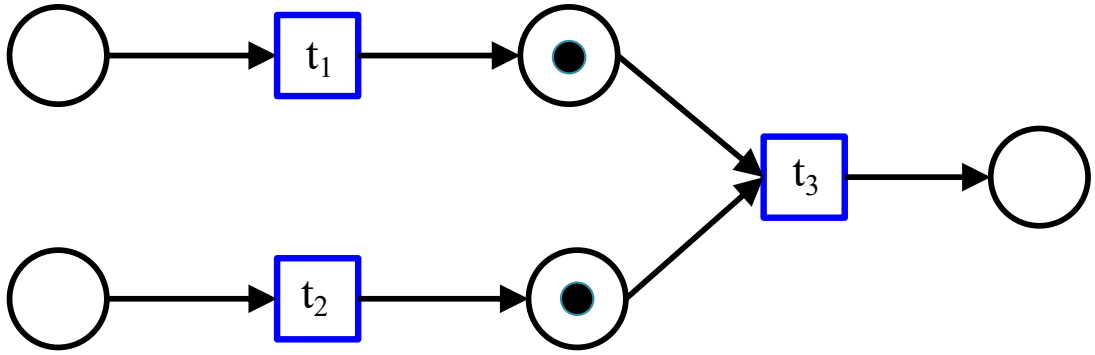


Forward execution: $\langle t_1, t_2, t_3 \rangle$

Reverse execution: $\langle t_3, t_2, t_1 \rangle$

Reversibility types – Backtracking

- Rewinding the computation trace
  - Exact reverse order
- Example

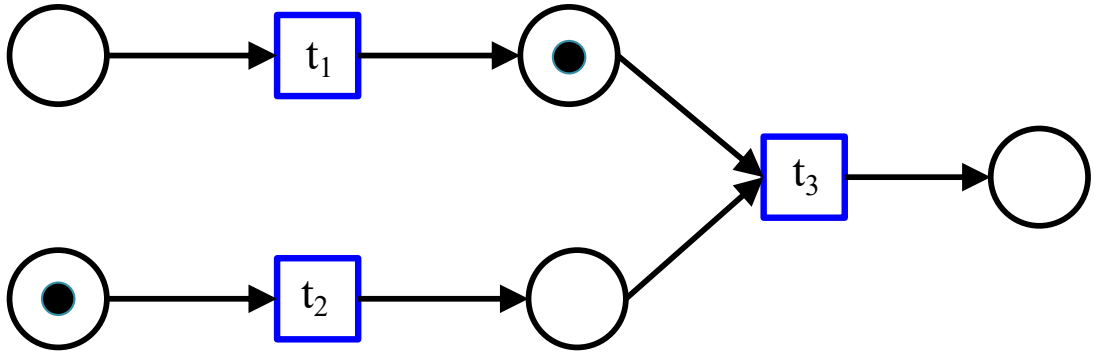


Forward execution: $\langle t_1, t_2, t_3 \rangle$

Reverse execution: $\langle t_3, t_2, t_1 \rangle$

Reversibility types – Backtracking

- Rewinding the computation trace
  - Exact reverse order
- Example

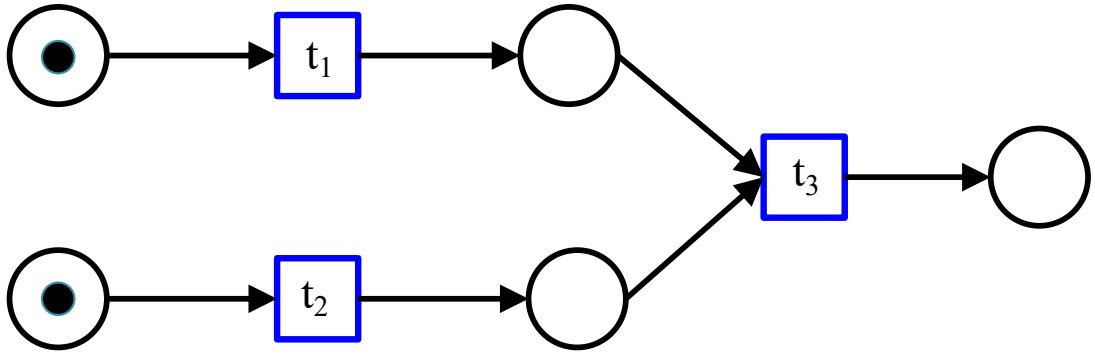


Forward execution: $\langle t_1, t_2, t_3 \rangle$

Reverse execution: $\langle t_3, t_2, t_1 \rangle$

Reversibility types – Backtracking

- Rewinding the computation trace
  - Exact reverse order
- Example



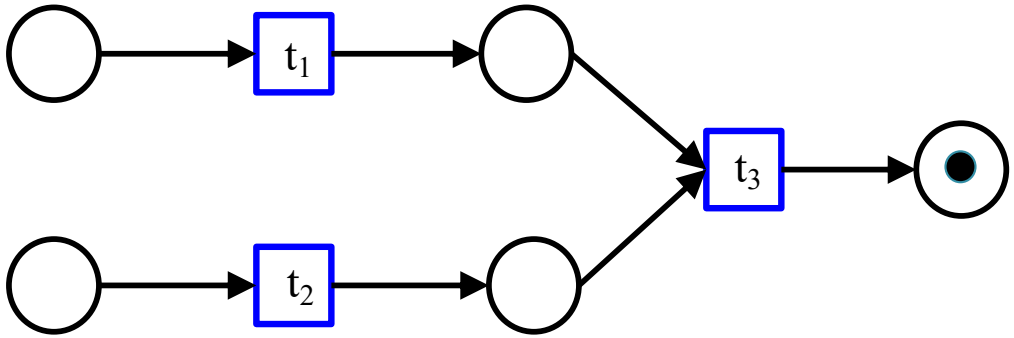
Forward execution: $\langle t_1, t_2, t_3 \rangle$

Reverse execution: $\langle t_3, t_2, t_1 \rangle$

- To implement backtracking one needs to *remember* the *history* of the execution

Reversibility types – Causal-order reversibility

- An event can reverse assuming all its effects have been reversed beforehand
  - Effects must be undone before their cause
  - More flexible form of reversibility than backtracking
- Example



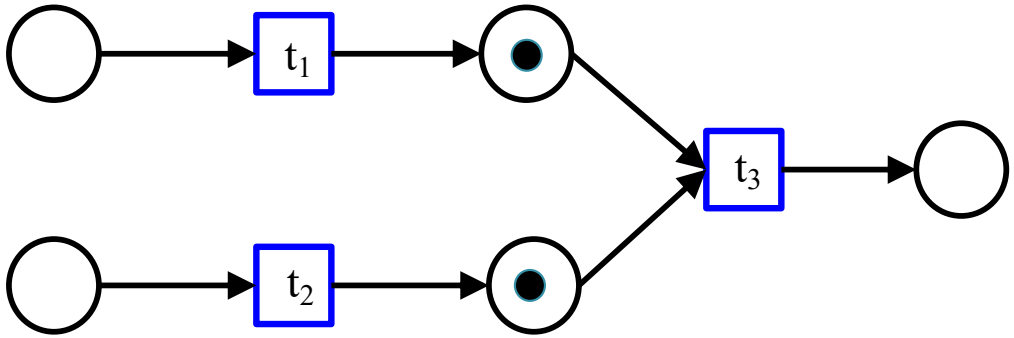
Forward execution: $\langle t_1, t_2, t_3 \rangle$

Reverse execution 1: $\langle t_3, t_2, t_1 \rangle$

Reverse execution 2: $\langle t_3, t_1, t_2 \rangle$

Reversibility types – Causal-order reversibility

- An event can reverse assuming all its effects have been reversed beforehand
  - Effects must be undone before their cause
  - More flexible form of reversibility than backtracking
- Example



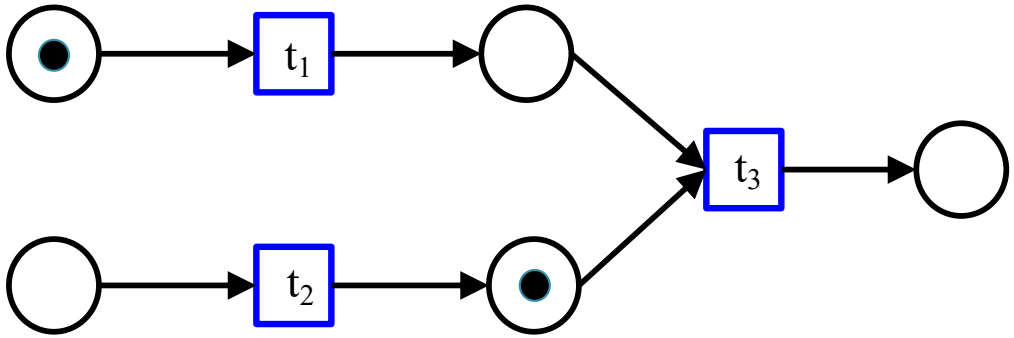
Forward execution: $\langle t_1, t_2, t_3 \rangle$

Reverse execution 1: $\langle t_3, t_2, t_1 \rangle$

Reverse execution 2: $\langle t_3, t_1, t_2 \rangle$

Reversibility types – Causal-order reversibility

- An event can reverse assuming all its effects have been reversed beforehand
  - Effects must be undone before their cause
  - More flexible form of reversibility than backtracking
- Example



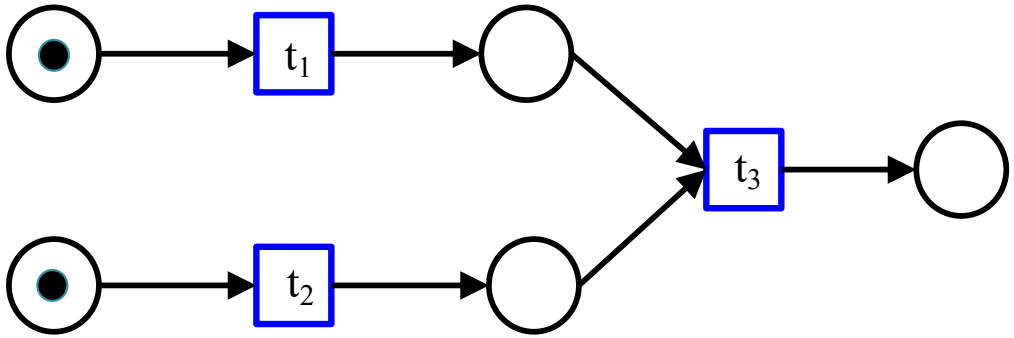
Forward execution: $\langle t_1, t_2, t_3 \rangle$

Reverse execution 1: $\langle t_3, t_2, t_1 \rangle$

Reverse execution 2: $\langle t_3, t_1, t_2 \rangle$

Reversibility types – Causal-order reversibility

- An event can reverse assuming all its effects have been reversed beforehand
  - Effects must be undone before their cause
  - More flexible form of reversibility than backtracking
- Example



Forward execution: $\langle t_1, t_2, t_3 \rangle$

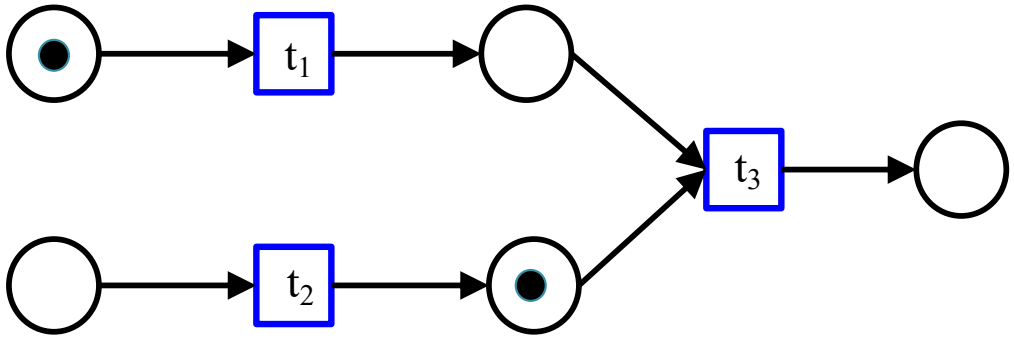
Reverse execution 1: $\langle t_3, t_2, t_1 \rangle$

Reverse execution 2: $\langle t_3, t_1, t_2 \rangle$

- During causal reversibility it is possible to reach states that did not occur previously

Reversibility types – Causal-order reversibility

- An event can reverse assuming all its effects have been reversed beforehand
  - Effects must be undone before their cause
  - More flexible form of reversibility than backtracking
- Example



Forward execution: $\langle t_1, t_2, t_3 \rangle$

Reverse execution 1: $\langle t_3, t_2, t_1 \rangle$

Reverse execution 2: $\langle t_3, t_1, t_2 \rangle$

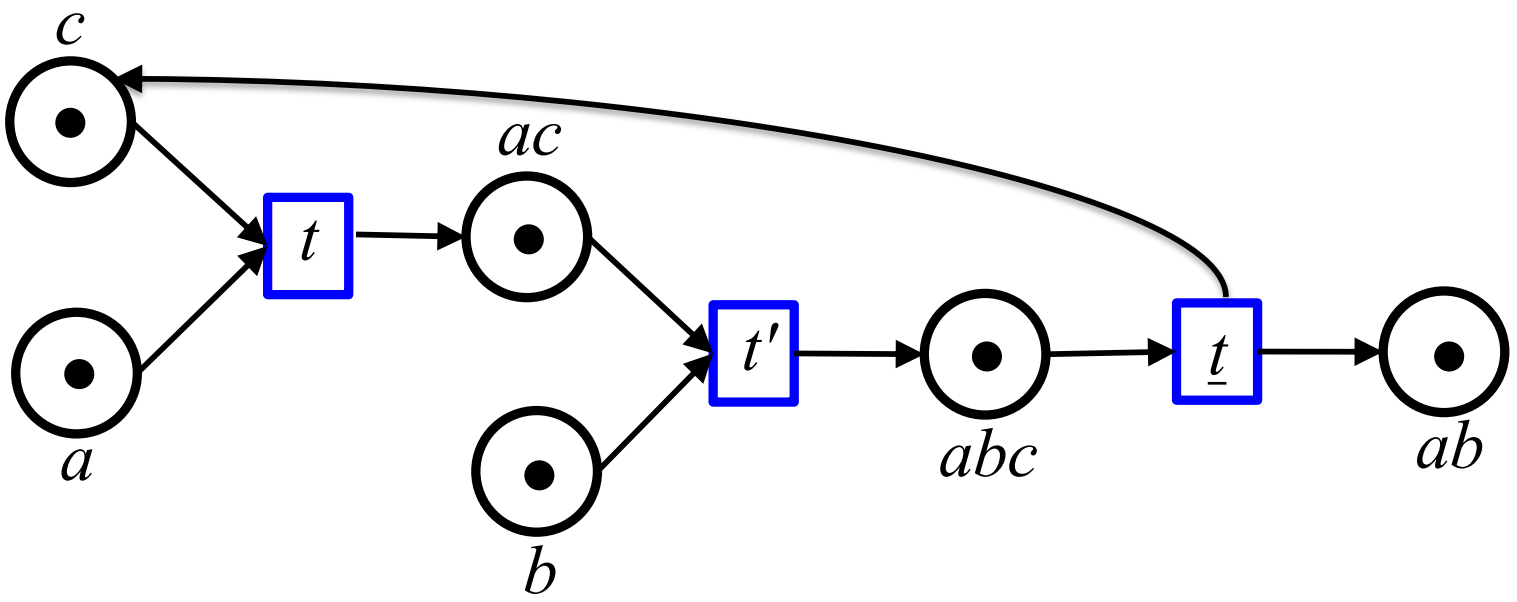
- During causal reversibility it is possible to reach states that did not occur previously
- To implement causal reversibility, one needs *to keep track of causal dependencies*

Reversibility types – Out-of-causal-order reversibility

- Allows undoing actions in an out-of-causal order
  - Inherent in systems such as biochemical reactions
  - Desirable in applications such as system reliability
- Example: Bonding of molecules A and B through the aid of catalyst C

- The third step is the reversal of the first step
  - It takes place in out-of-causal order
  - It creates a fresh, otherwise unreachable, state
- To implement out-of-causal-order reversibility, one needs *to keep track of the effect of transitions*

Catalysis example in classical PNs

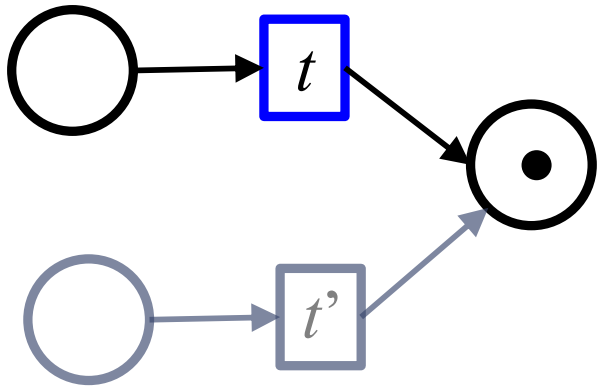


Transition $\underline{t}$ represents the reversal of t

Aim: Propose a model where reversibility is modeled directly – without the need of additional transitions

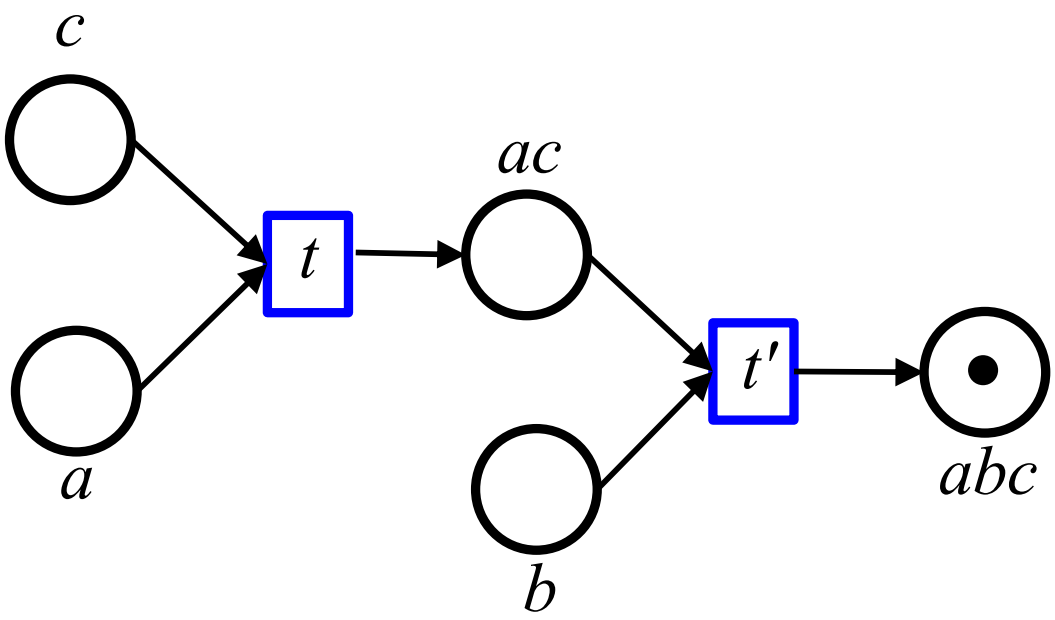
Reversing computation in PNs – challenges

- What does one need to remember in order to reverse transitions in PNs?
- How do we identify legitimate backward moves?
- How does one deal with backward nondeterminism?



Reversing computation in PNs – challenges

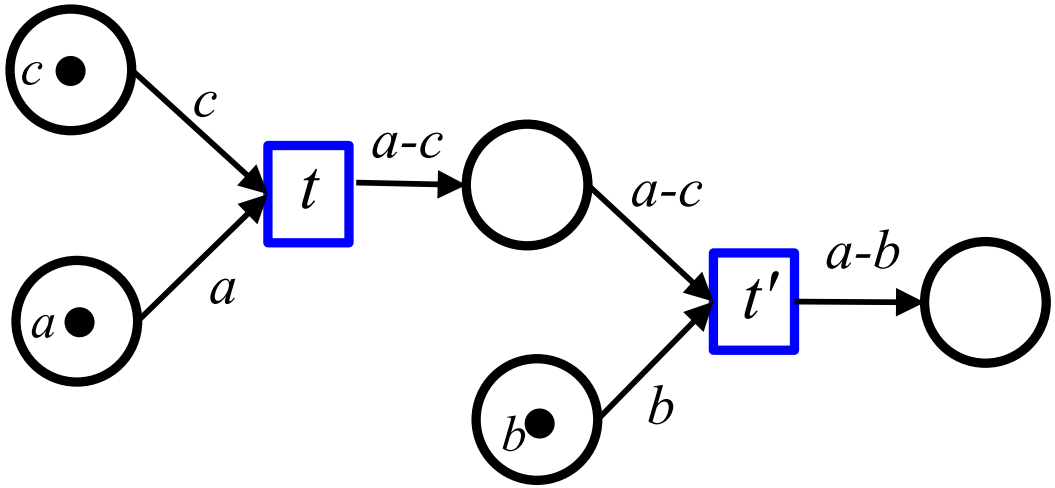
- How do we reverse the effect of a transition in out-of-causal order?



RPNs – Design decisions

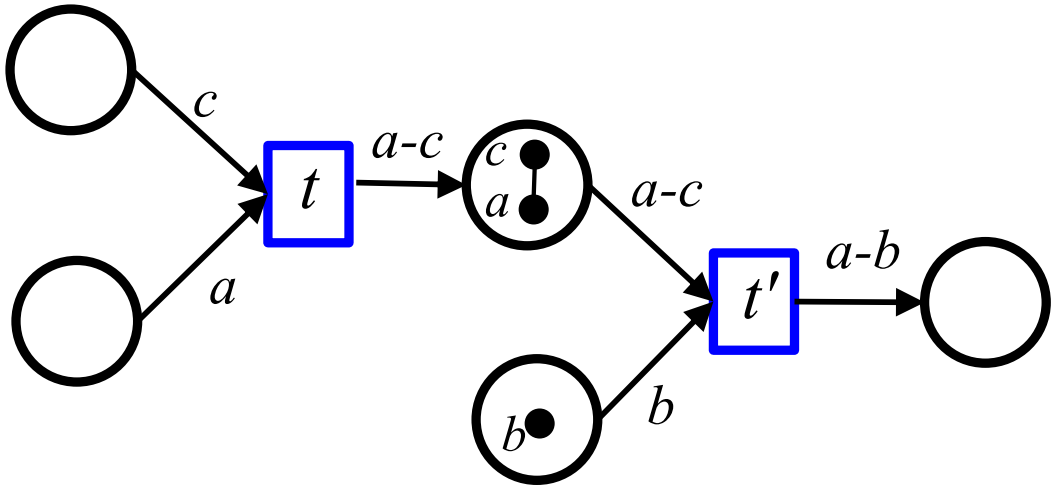
- Individual tokens
  - Each token is identified by its name/type: *a*, *b*, …
  - Tokens are preserved
  - Tokens can be bonded together: *a*-*b*, *b*-*c*, …
- Transitions
  - Move tokens from incoming to outgoing places
  - May form bonds between tokens
- Transition effect: the created bonds
  - Reversing a transition involves undoing the bonds created by the transition
- Histories
  - Transitions are associated with keys, which capture the order in which transitions were executed
  - Convey information about causal paths
- RPNs in forward-only semantics: High-level Petri nets with individual tokens that satisfy the conservativeness property, where functions form bonds

Catalysis in RPNs



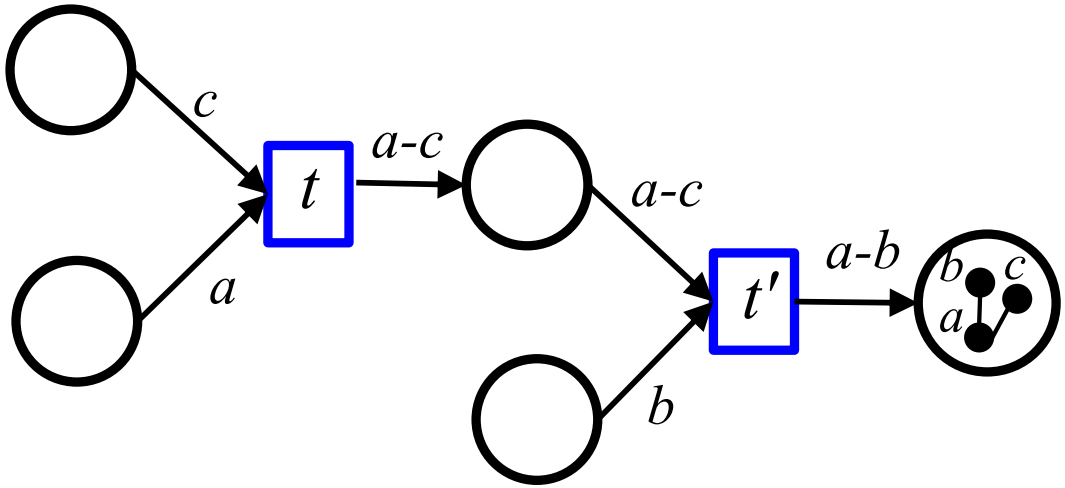
Execute *t*: form bond between *a* and *c*

Catalysis in RPNs



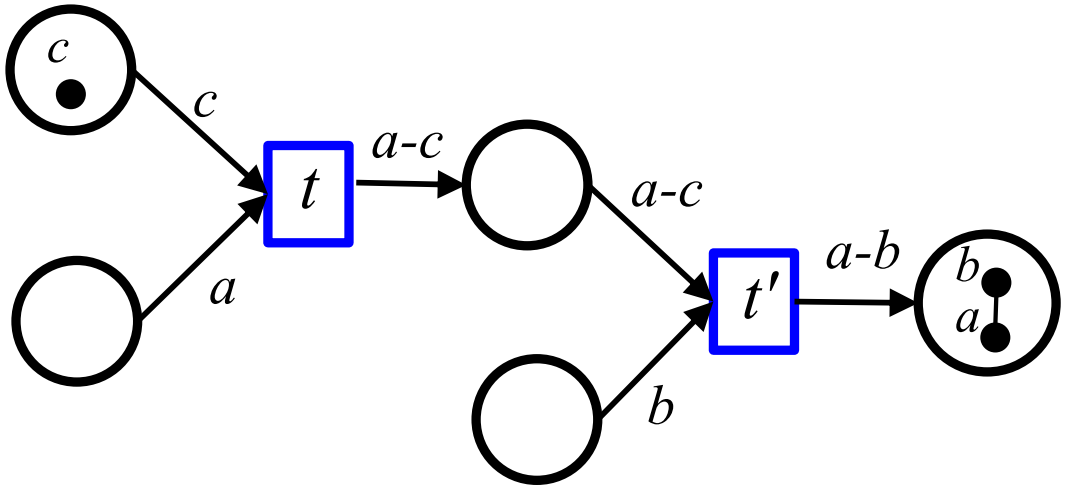
Execute *t'*: form bond between *a* and *b*

Catalysis in RPNs



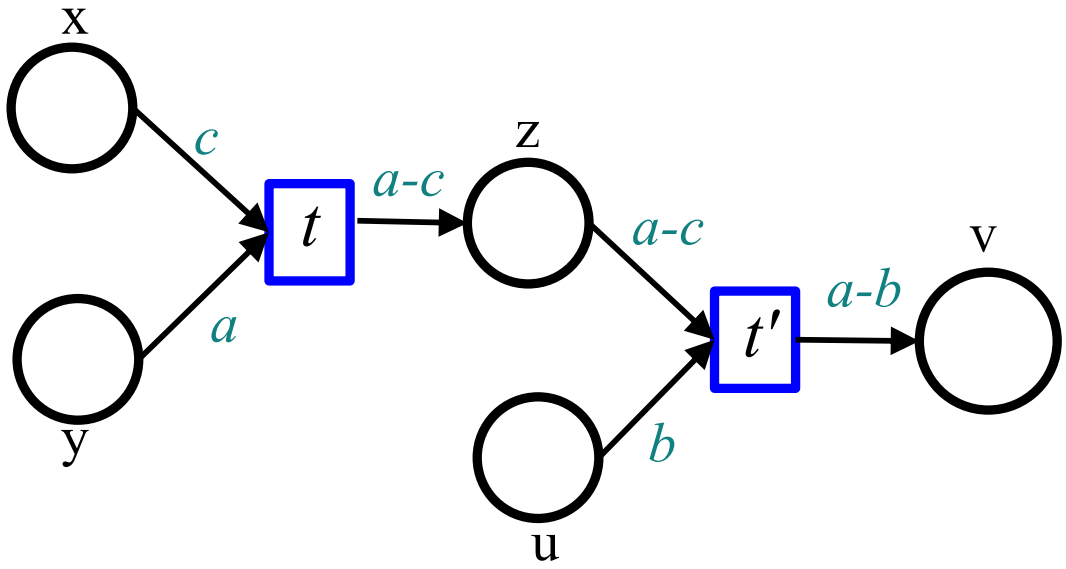
Reverse *t*: release *c* from the bond with *a*

Catalysis in RPNs



Reversing Petri nets - definition

- A Reversing Petri net is a tuple (P, T, A, B, F) where:
  - P is a finite set of places
  - T is a finite set of transitions
  - A is a finite set of bases or tokens
  - $B \subseteq A \times A$ is a set of bonds
  - $F : (P \times T \cup T \times F) \to 2^{A \cup B}$ is a set of directed arcs

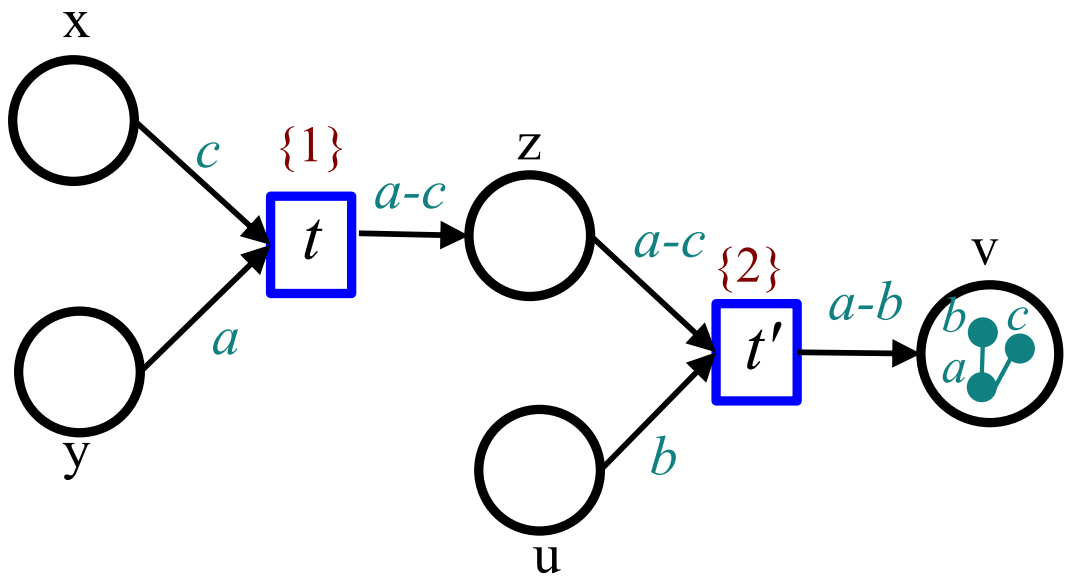


A. Philippou and K. Psara. *Reversible computation in Petri nets*. In Proceedings of RC 2018, LNCS 11106, pages 84–101. Springer, 2018

A. Philippou and K. Psara. *Reversible computation in nets with bonds.* J. Log. Algebraic Methods Program. 124: 100718 (2022)

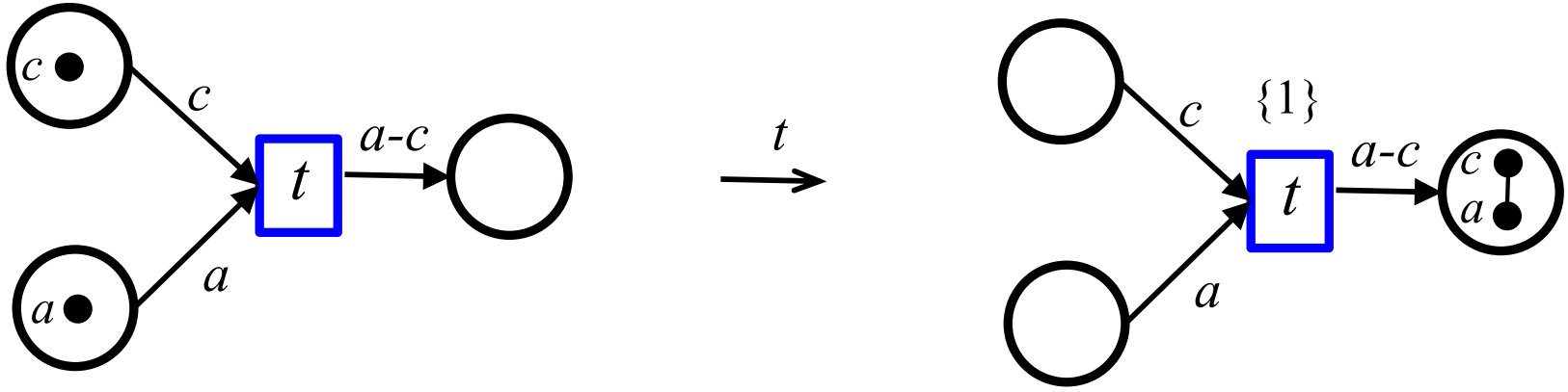
Reversing Petri nets – state

- RPN state: $\langle M, H \rangle$
  - ***M***, the marking, is an assignment of tokens and bonds across the places of the RPN
  - ***H***, the history, is an assignment of keys to each transition that capture the execution sequence



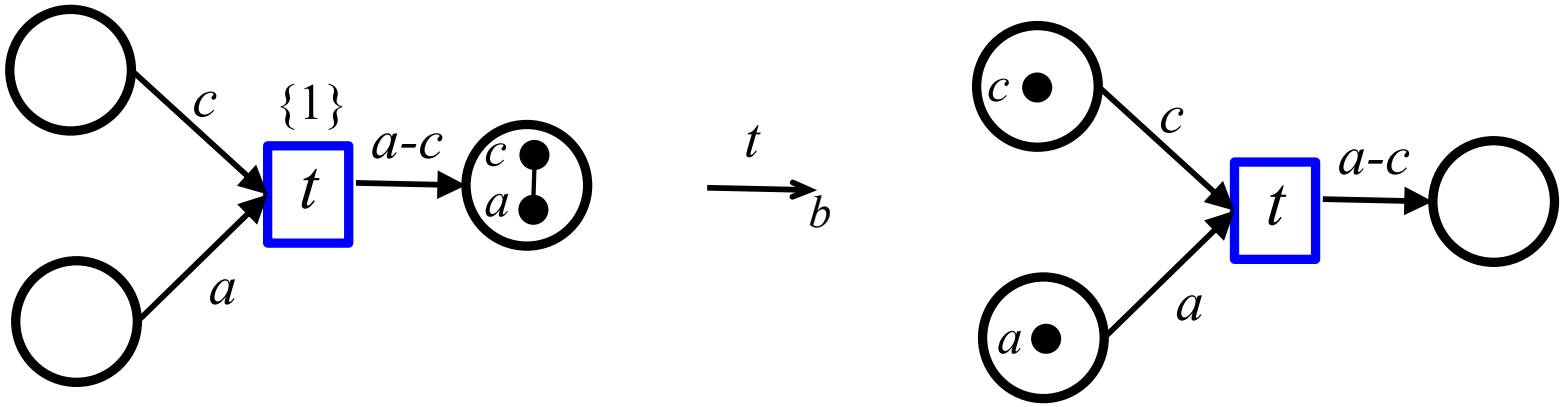
Forward execution

- Forward execution: $\langle M, H \rangle \xrightarrow{t} \langle M', H' \rangle$
  - A transition may be executed if the required tokens are available
  - Tokens and their connected components are transferred to the outgoing places of the transition
  - New bonds can be created
  - History information is updated



Backtracking execution

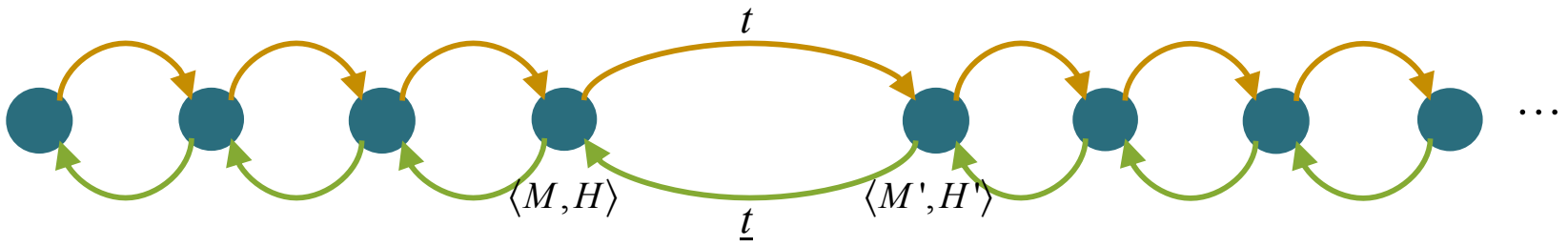
- Backtracking: $\langle M, H \rangle \xrightarrow{t}_b \langle M', H' \rangle$
  - The last executed transition can be reversed (based on H)
  - Tokens are moved from the outgoing places of the transition to its incoming places
  - Bonds created by the transition are broken
  - History information is updated



Backtracking execution

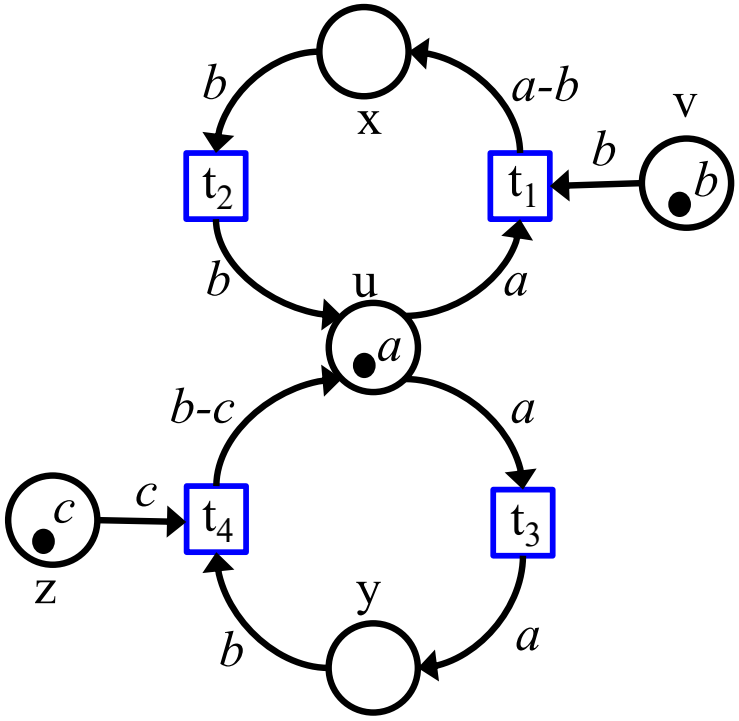
Loop Lemma

For any forward transition $\langle M, H\rangle \xrightarrow{t} \langle M', H'\rangle$ there is a backtracking transition $\langle M', H'\rangle \xrightarrow{t}_b \langle M, H\rangle$, and vice versa.



Causal-order reversibility

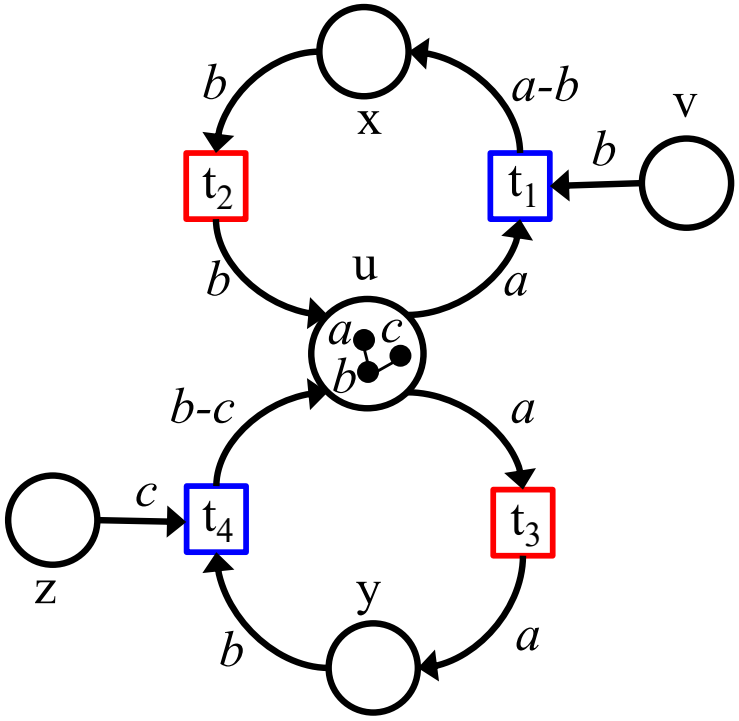
- Causal-order reversing: $\langle M, H \rangle \xrightarrow{t}_{c} \langle M', H' \rangle$
  - A transition can be reversed if all transition occurrences it has caused have already been reversed
  - Tokens are moved from the outgoing places of the transition to its incoming places
  - Bonds created by the transition are broken



- A causal link exists between two transitions if one produces tokens used to fire the other

Causal-order reversibility

- Causal-order reversing: $\langle M, H \rangle \xrightarrow{t}_{c} \langle M', H' \rangle$
  - A transition can be reversed if all transition occurrences it has caused have already been reversed
  - Tokens are moved from the outgoing places of the transition to its incoming places
  - Bonds created by the transition are broken

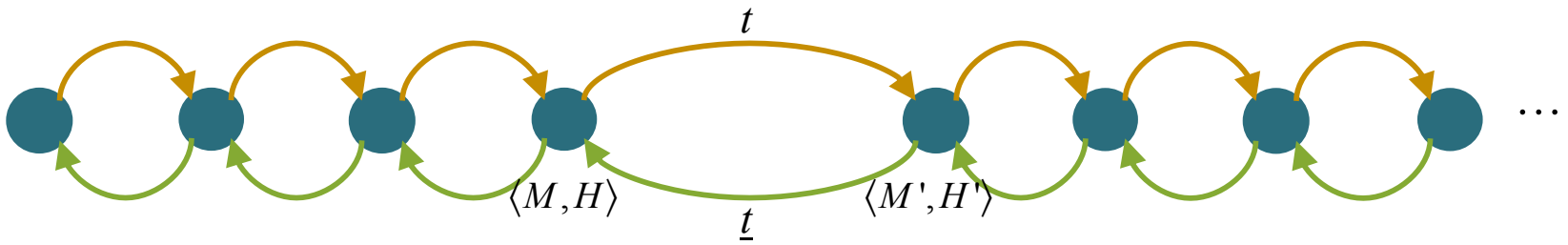


- A causal link exists between two transitions if one produces tokens used to fire the other
  - After execution of t_1,t_2,t_3,t_4 it is not possible to reverse t_2 since it has caused t_3 and t_4
- Cannot be determined merely on structural information – need to refer to the state
  - A causal dependence relation is constructed while computation proceeds

Causal-order reversibility

Loop Lemma

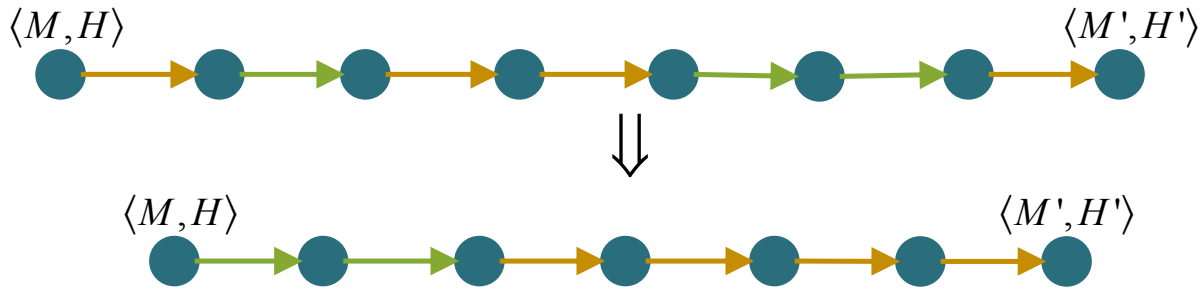
For any forward transition $\langle M, H\rangle \xrightarrow{t} \langle M', H'\rangle$ there is a causal-order reverse transition $\langle M', H'\rangle \xrightarrow{t}_{c} \langle M, H\rangle$, and vice versa.



Causal-order reversibility

Parabolic Lemma

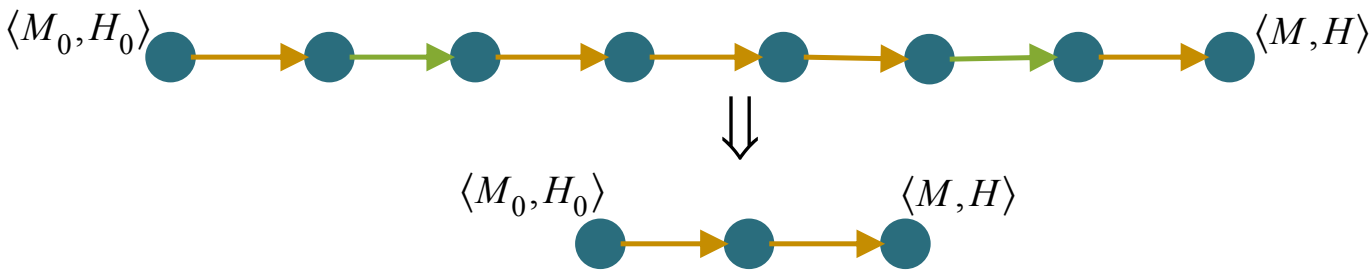
For any execution $\langle M, H\rangle \xrightarrow{\sigma}_c \langle M', H'\rangle$, where σ is a sequence of both forward and reverse transitions, there exists $\langle M, H\rangle \xrightarrow{\underline{r}}_c \langle M', H'\rangle \xrightarrow{r'}_c M', H'\rangle$, where $\underline{r}$ is a sequence of reverse actions and r' a sequence of forward actions.



Causal-order reversibility

Corollary

For any execution $\langle M_0, H_0\rangle \xrightarrow{\sigma}_c \langle M, H\rangle$, where $\langle M_0, H_0\rangle$ is the initial state and σ is a sequence of both forward and reverse transitions, there exists a forward-only execution σ' such that $\langle M_0, H_0\rangle \xrightarrow{\sigma'}_c \langle M, H\rangle$.



Causal-order reversibility

Causal Consistency Theorem

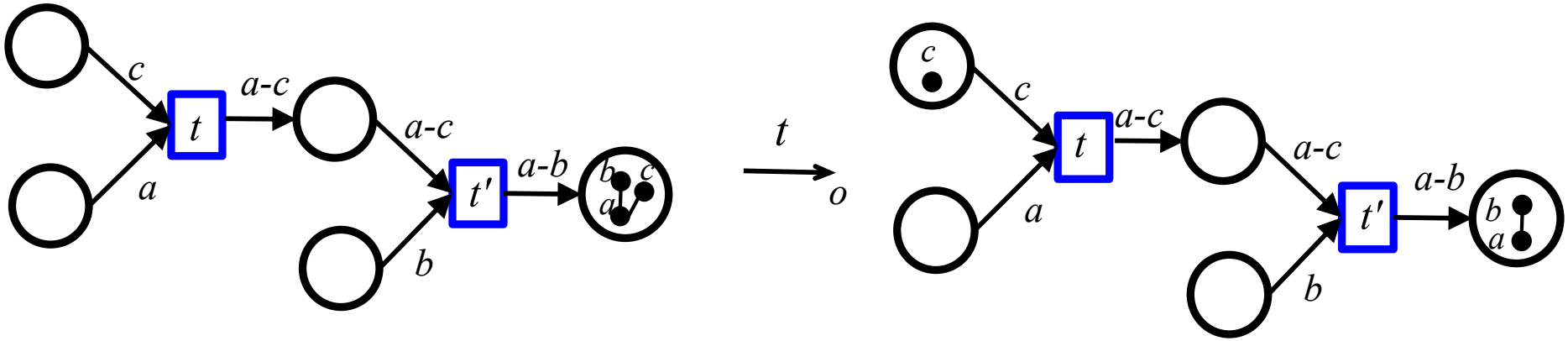
Two sequences of transitions lead to the same states

$$\langle M, H \rangle \xrightarrow{\sigma_1}_c \langle M', H' \rangle \text{ and } \langle M, H \rangle \xrightarrow{\sigma_2}_c \langle M', H' \rangle,$$

if and only if σ_1 and σ_2 differ only by reordering of *independent* transitions and inserting or removing pairs of *opposite* actions.

Out-of-causal-order reversibility

- Out-of-causal order reversing: $\langle M, H \rangle \xrightarrow{t}_{o} \langle M', H' \rangle$
  - Any executed transition can be reversed
  - Bonds created by the transition are broken
  - Tokens are transferred to the place they would have occurred without the effect of the reversed transition



- May give rise to states that are not reachable by forward-only execution
  - Nonetheless the states are uniquely characterized by the effects of the non-reversed transitions

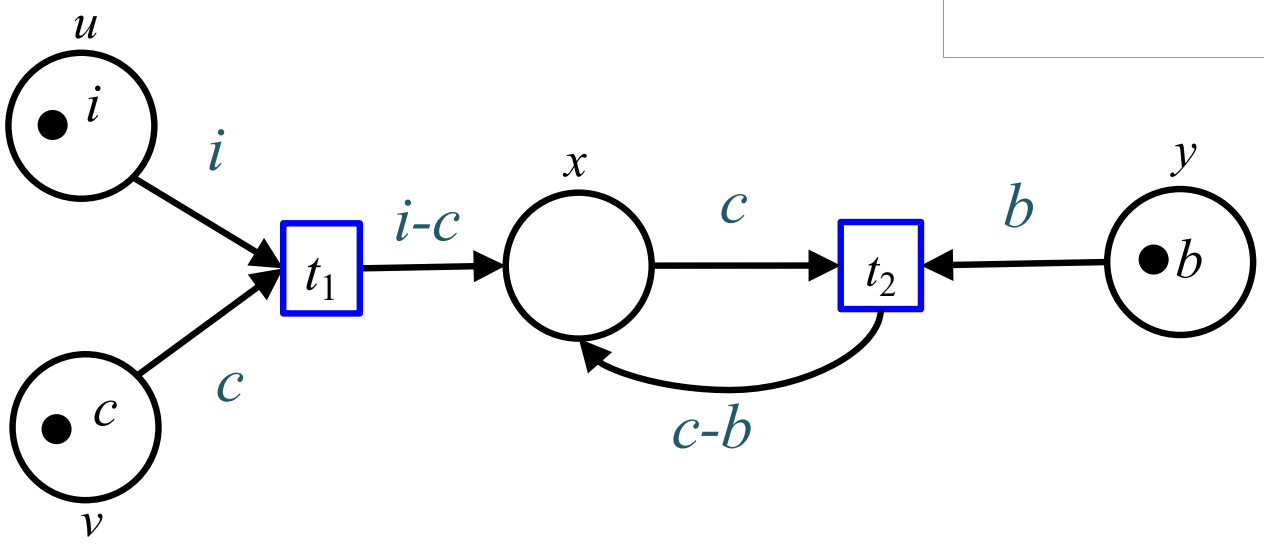
Relation between reversibility types

Theorem $\rightarrow_b \subset \rightarrow_c \subset \rightarrow_o$

Reversing Petri nets – Multi-token RPNs and the Individual-Token Interpretation

Example – Pen assembly/disassembly

- System with three components:
  - The ink, *i*
  - The cup, *c*
  - The button, *b*

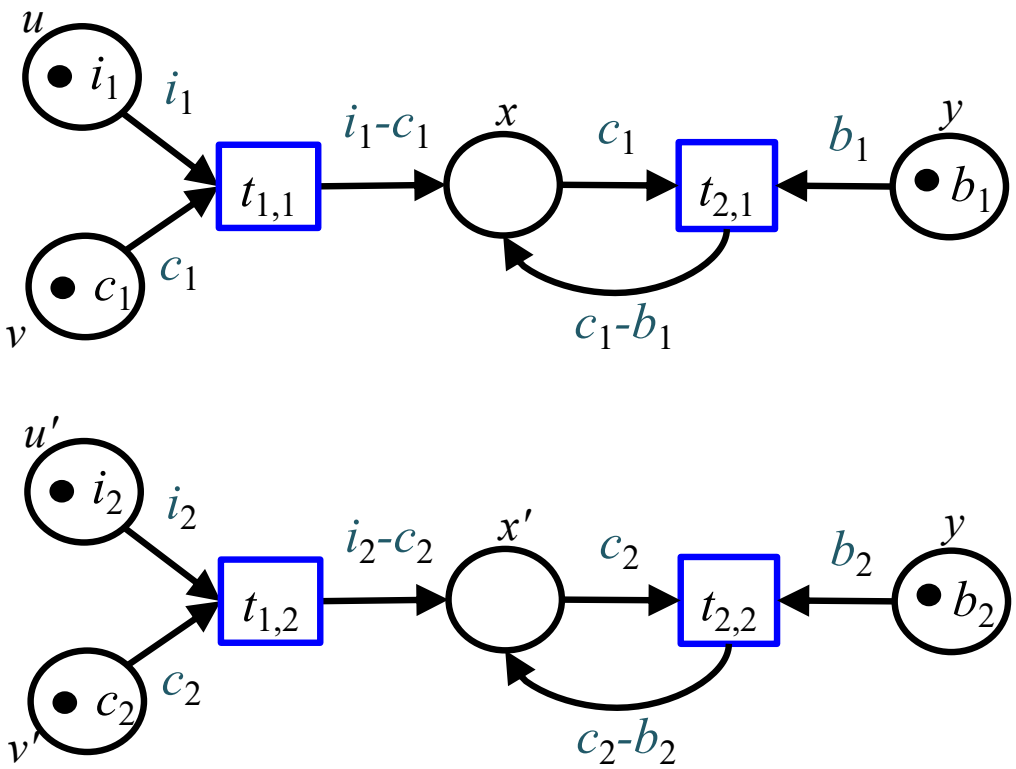


The bond *i-c* captures that the ink is in the cup and the bond *c-b* captures that the button is screwed on the cup

Two pens?

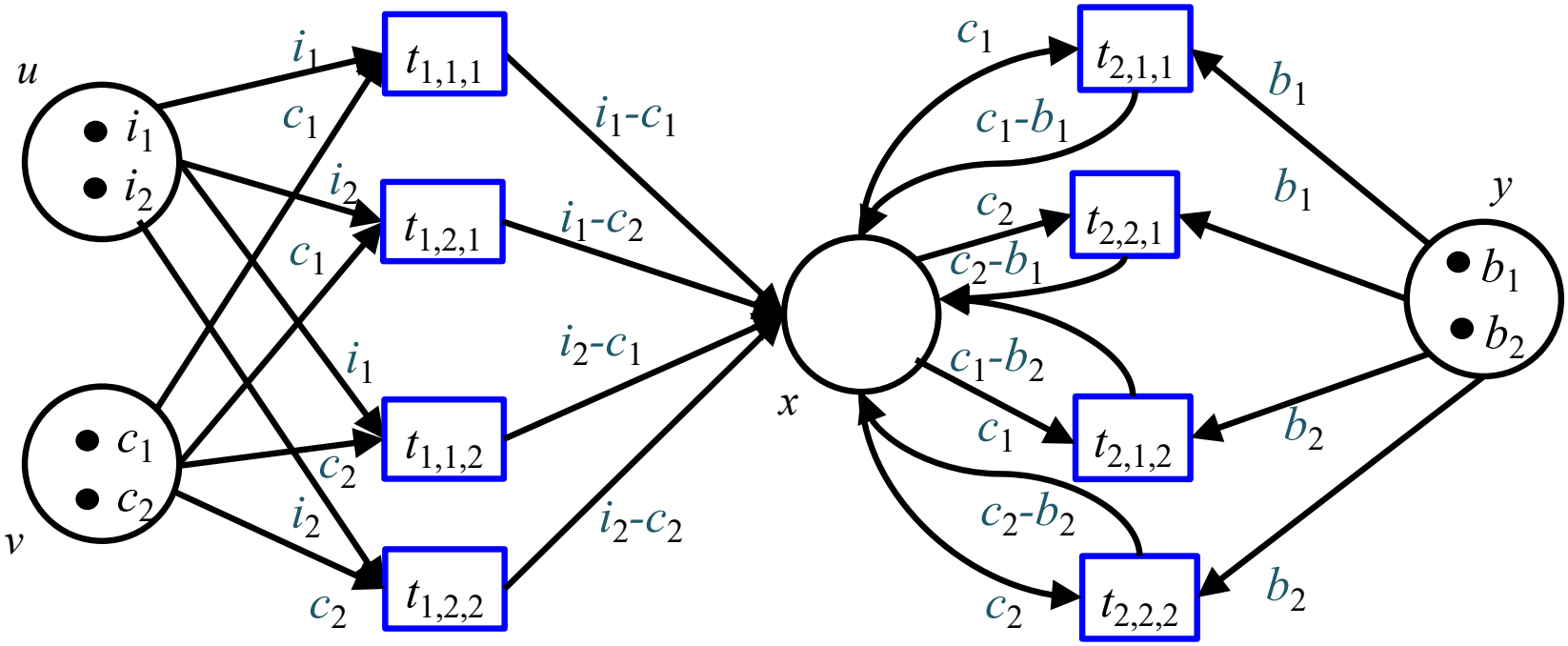
- RPNs feature named tokens that are pairwise distinct
  - There exists exactly one token of each “type”
- How can we model a system with two pens?
- RPN with six tokens:
  - The inks, i_1, i_2
  - The cups, c_1, c_2
  - The buttons, b_1, b_2

Two pens?

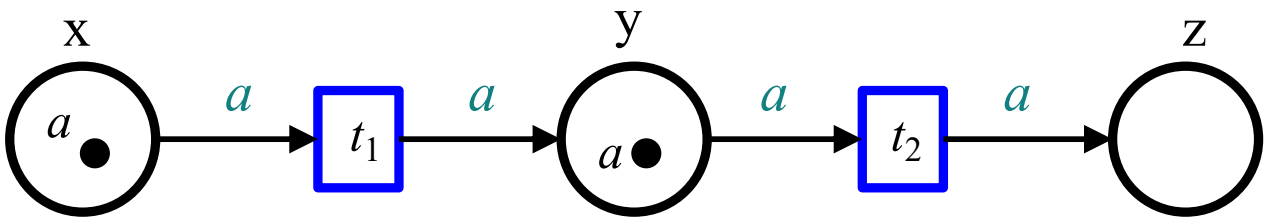


But any combination of assemblies should be possible - tokens of the same type should be indistinguishable

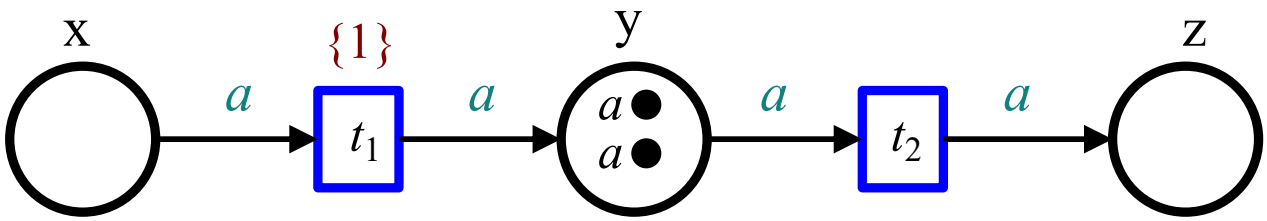
Two pens


u
v
x
y
i_1
i_2
c_1
c_2
b_1
b_2
$t_{1,1,1}$
$t_{1,2,1}$
$t_{1,1,2}$
$t_{1,2,2}$
$t_{2,1,1}$
$t_{2,2,1}$
$t_{2,1,2}$
$t_{2,2,2}$
i_1-c_1
i_1-c_2
i_2-c_1
i_2-c_2
c_1-b_1
c_2-b_1
c_1-b_2
c_2-b_2

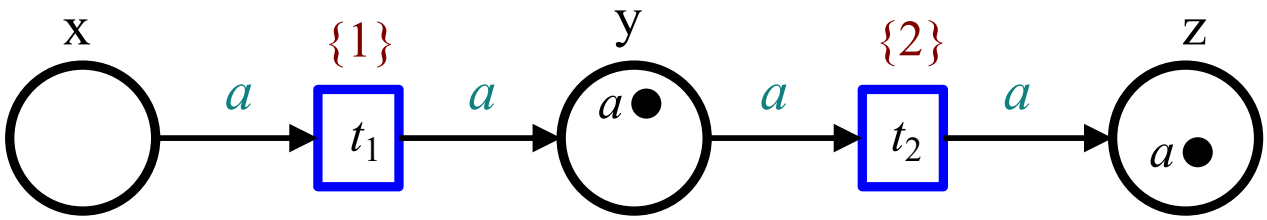
Introducing multiple tokens



Introducing multiple tokens



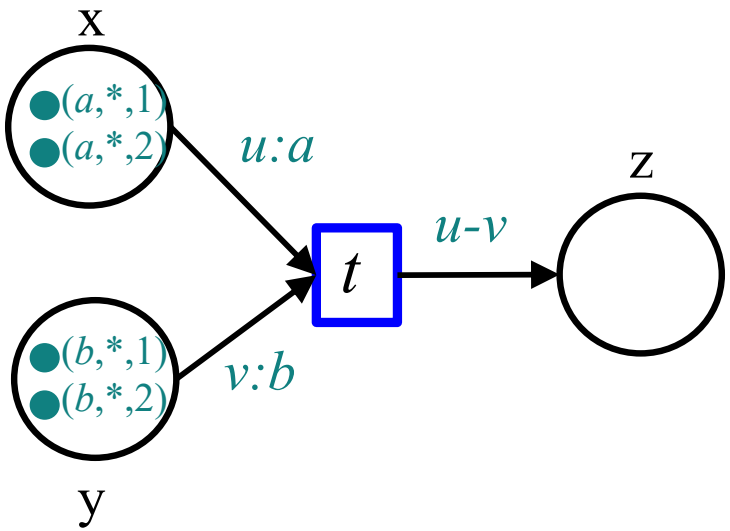
Introducing multiple tokens



- Has t_1 caused t_2? In other words, has the token that fired t_1 also fired t_2?
- Two approaches to token multiplicity
  - Individual-token interpretation: each token is considered unique and should be identified by its causal path
  - Collective-token interpretation: tokens of the same type are considered as identical
- Is it possible to reverse t_1?
  - Individual-token interpretation: It depends on which a was used to fire t_2.
  - Collective-token interpretation: Yes (as long as there is an available token)

Individual-token interpretation

- Token types
  - Multiple *instances* of a token type may exist in a net
  - Token instances of the same type have the same capabilities
- Arcs are associated with typed variables
  - u:A : request for a token of type A
- Tokens have a memory of their causal path
  - $((\ldots((a,k_1,v_1), k_2,v_2)\ldots), k_n,v_n)$:

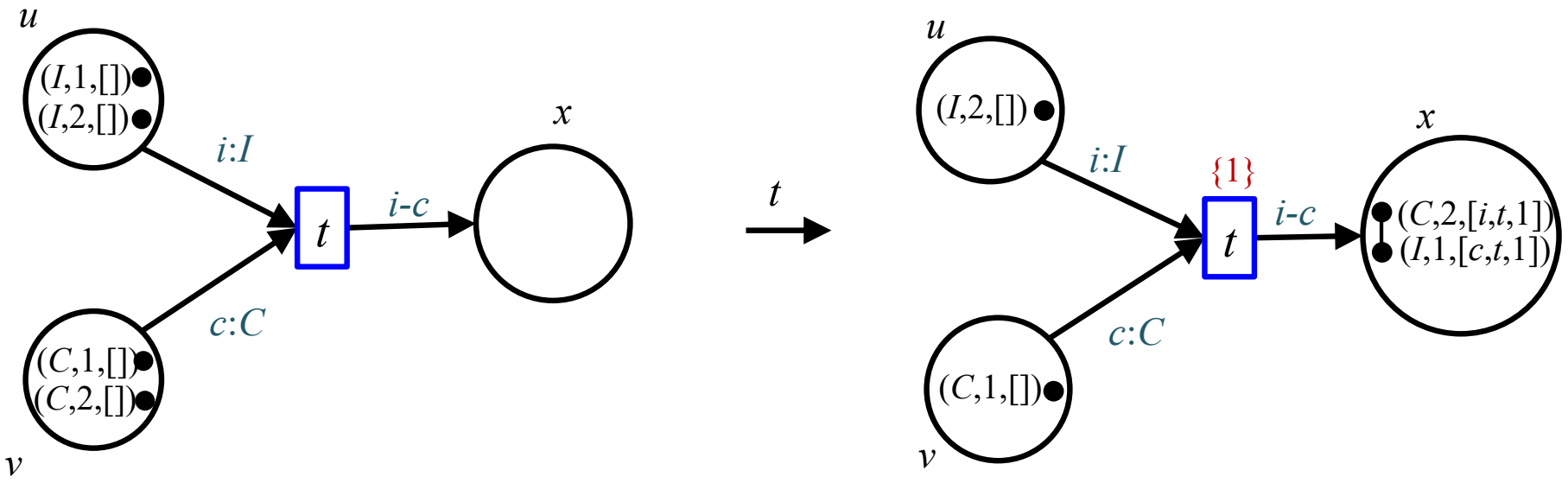


token a has participated in n transitions with keys $k_1,..,k_n$ and forming variable associations with $v_1,...,v_n$

- All three forms of reversibility are supported
  - Information needed for transition reversal is implemented "locally" at the token level

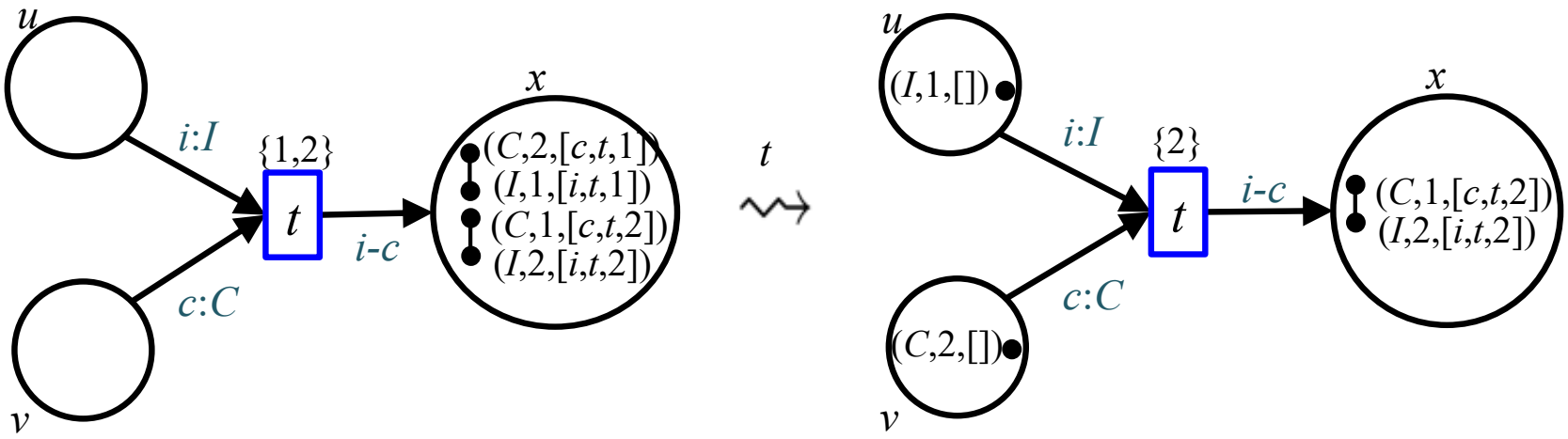
Forward execution

- A transition is enabled if:
  - There is a collection of token/bond instances in the incoming places of the transition that can be instantiated to the incoming variables of the transition
- Firing a transition results in:
  - Transferring all relevant tokens from the incoming places to the outgoing places and creating/destroying bonds as specified by the transition
  - Extending the history of the transition with the next available key in ascending order
  - Updating the causal path of the tokens involved in the newly-executed transition



Causal-order reversing

- A causal link exists between two transitions if one produces tokens used to fire the other
- A transition occurrence can be reversed if:
  - All the tokens/bond instances involved in firing the occurrence have not engaged in any further transitions or, if they did, these transitions have been reversed
- Reversing a transition results in:
  - Transferring all relevant token/bond instances from the outgoing places to the incoming places forming/breaking bonds as necessary
  - The history of the transition is updated by removing the key of the reversed transition occurrence
  - Updating the causal path of the tokens by removing the record of the reversed transition



Individual-token interpretation

Theorem

Under the individual-token interpretation, for each RPN with multiple tokens there exists an equivalent RPN with single tokens, and vice versa.

Proof idea

It is possible to construct a translation from RPNs with multiple tokens to RPNs with a single token of each type. Their equivalence manifests itself as an isomorphism of reachable parts of the associated LTSs

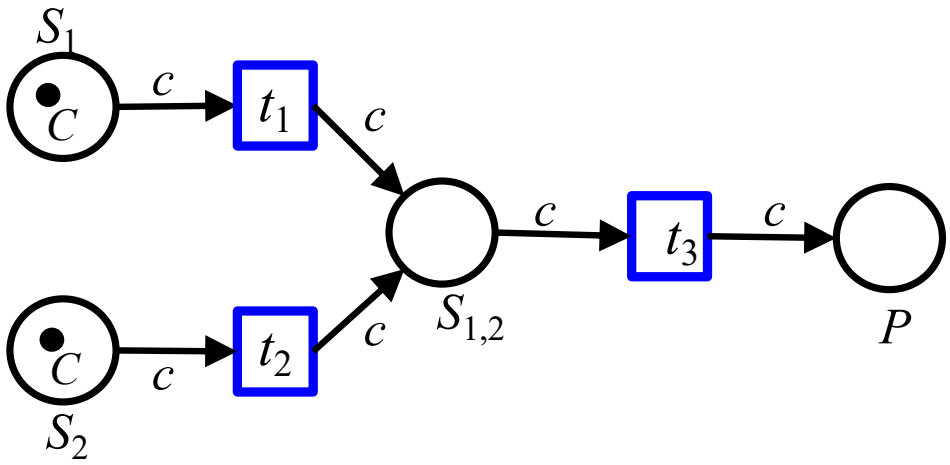
Conclusions

1. Global information regarding state/causal dependences in the original RPN model can be captured locally as histories in tokens
2. Multiple tokens do not increase the expressiveness of the model

Reversing Petri nets – Multi-token RPNs and the Collective-Token Interpretation

Collective-token interpretation

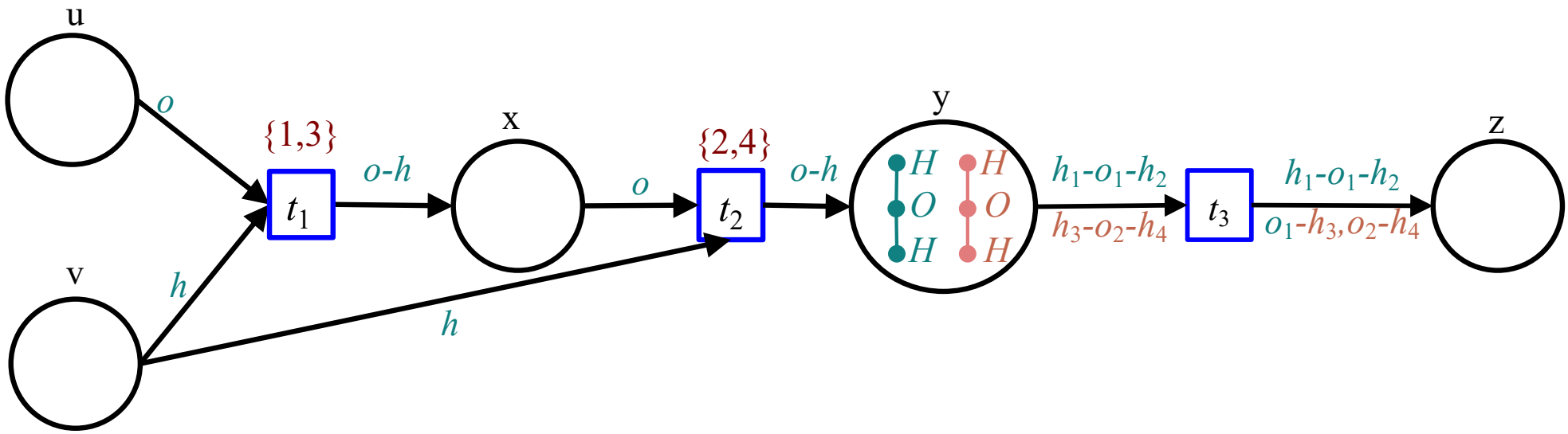
- Tokens of the same type are not distinguished
  - Philosophy maintained in various application domains, e.g. recourse aware systems or biological systems
  - Enables an alternative reversibility type closely related to disjunctive causality



- No global control
  - Transitions are reversed based on local conditions

Autoprotolysis of water

- Two identical water molecules H_20 transfer a hydrogen atom H to become hydronium H_30 and hydroxide H0

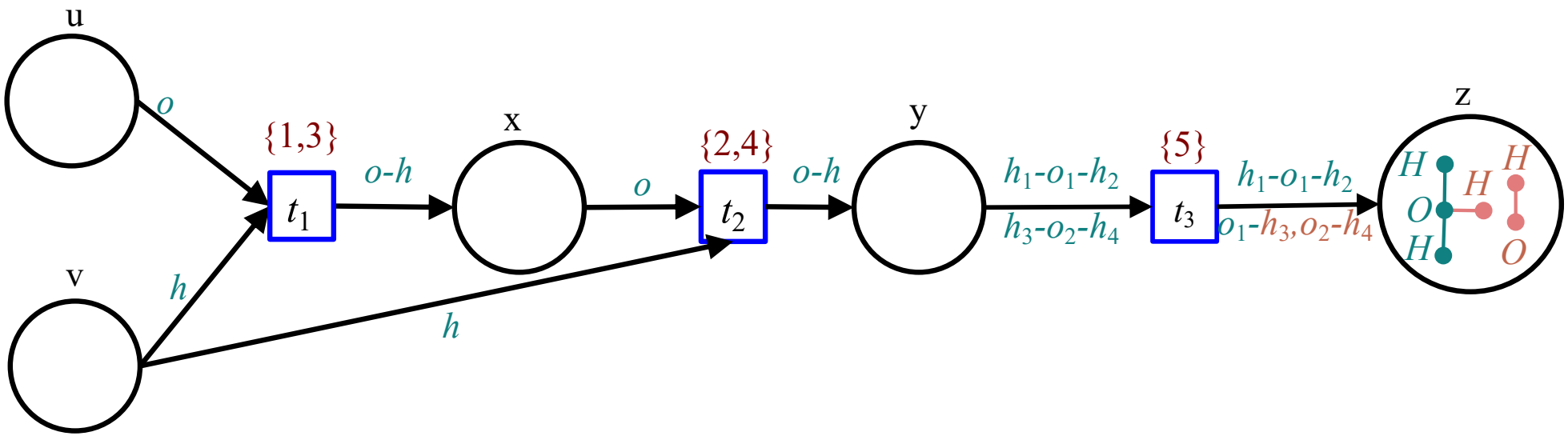


- An oxygen in a water molecule can become attracted by a hydrogen in another water molecule due to their opposite charges

S. Kuhn, B. Aman, G. Ciobanu, A. Philippou, K. Psara, and I. Ulidowski. *Reversibility in Chemical Reactions*. In Reversible Computation: Extending Horizons of Computing – Selected Results of the COST Action IC1405 (pp. 151-176). LNCS 12070. Springer, 2020

Autoprotolysis of water

- Two identical water molecules H_20 transfer a hydrogen atom H to become hydronium H_30 and hydroxide H0



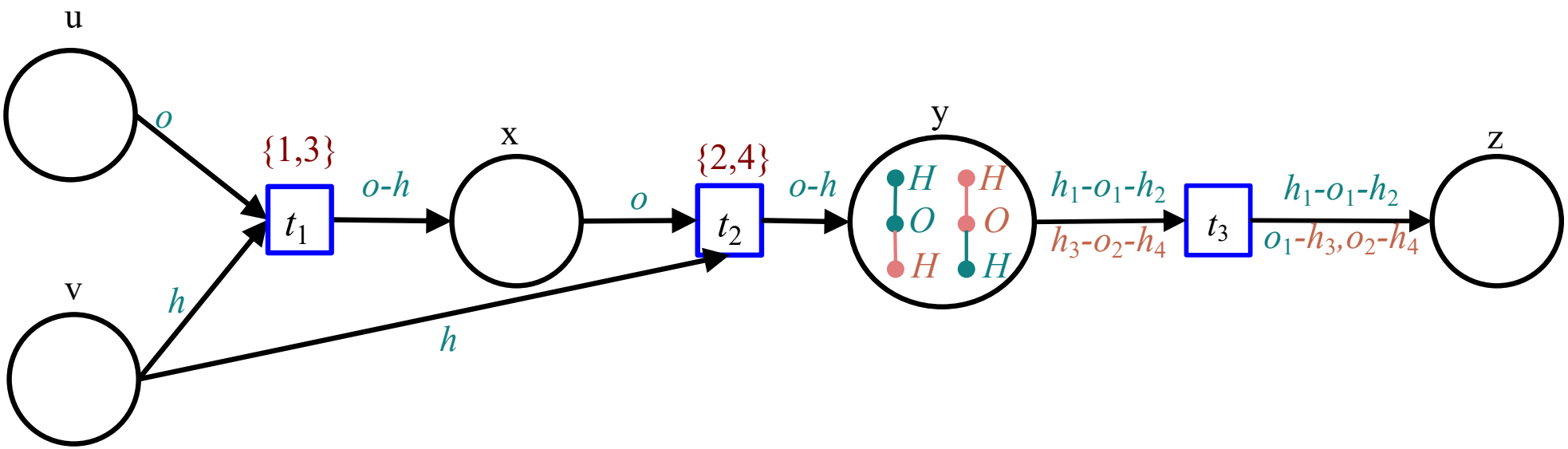
- Since a hydrogen atom cannot have more than one bond, the existing bond is broken

- The reaction is reversible - the different bonds cannot be distinguished

S. Kuhn, B. Aman, G. Ciobanu, A. Philippou, K. Psara, and I. Ulidowski. *Reversibility in Chemical Reactions*. In Reversible Computation: Extending Horizons of Computing – Selected Results of the COST Action IC1405 (pp. 151-176). LNCS 12070. Springer, 2020

Autoprotolysis of water

- Two identical water molecules H_20 transfer a hydrogen atom H to become hydronium H_30 and hydroxide H0



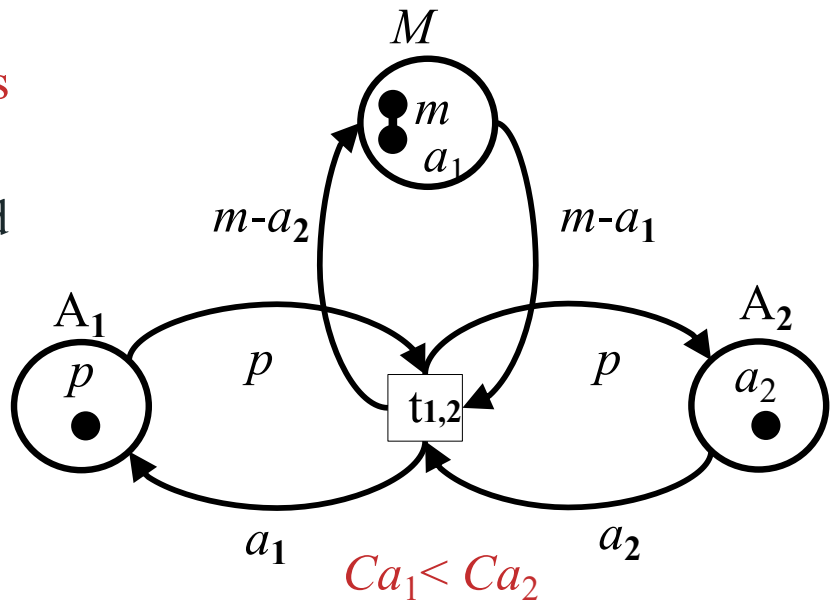
- Any of the hydrogens of the H_30 may be released and form a bond with the HO
- Model successfully captures *concerted* actions, i.e. actions that simultaneously create and break bonds.

S. Kuhn, B. Aman, G. Ciobanu, A. Philippou, K. Psara, and I. Ulidowski. *Reversibility in Chemical Reactions*. In Reversible Computation: Extending Horizons of Computing – Selected Results of the COST Action IC1405 (pp. 151-176). LNCS 12070. Springer, 2020

Controlling reversibility

- In practice, reversibility does not take place liberally
  - In fault-tolerant systems it should be applied when a fault is encountered
  - In biochemical systems it is triggered by environmental conditions

- Controlling reversibility in RPNs
  - Tokens are associated with data values
  - Transitions are associated with conditions whose satisfaction controls whether the transitions can be executed in the forward or the reverse direction



A. Philippou, K. Psara, and H. Siljak. *Controlling reversibility in reversing Petri nets with application to wireless communications*. In Proceedings of RC 2019, LNCS 11497, pages 238-245. Springer, 2019.

Conclusions

Applications

- Novel distributed algorithm for antenna selection in MIMO systems [1]
- Transaction-processing systems [2]
- Applications from Biochemistry
  - Autoprotolysis of water [3], the ERK signaling pathway [4], the Ammonium chloride chemical reaction [5]
- Further mechanisms for fine-tuning transition execution
  - Inhibitor arcs: inhibition is determined by the presence of bonds
  - The possibility of simultaneously creating and destroying bonds during a transition firing

[1] H. Siljak, K. Psara, K., and A. Philippou. *Distributed antenna selection for massive MIMO using reversing Petri nets*. IEEE Wireless Communication Letters 8, 5 (2019), 1427–1430.

[2] A. Philippou and K. Psara. *Reversible computation in Petri nets*. In Proceedings of RC 2018, LNCS 11106, pages 84–101. Springer, 2018

[3] S. Kuhn, B. Aman, G. Ciobanu, A. Philippou, K. Psara, and I. Ulidowski. *Reversibility in Chemical Reactions*. In Reversible Computation: Extending Horizons of Computing – Selected Results of the COST Action IC1405 (pp. 151-176). LNCS 12070. Springer, 2020.

[4] A. Philippou and K. Psara. *Reversible computation in cyclic Petri nets*. Under submission

[5] K. Psara. *Reversible computation in Petri nets*. PhD thesis, December 2020

On-going and future work

- Relationship between RPNs and Colored Petri nets [1]
- Tool development
  - Prototype simulator
  - Translation into ASP [2]
  - Model checking and analysis techniques
- Applications
- Further investigation of the concept of causal reversibility based on the notion of disjunctive causality
- Extension to quantum systems

[1] K. Barylska, A. Gogolinska, L. Mikulski, A. Philippou, M. Piatkowski, and K. Psara. *Reversing computations modelled by coloured Petri nets*. In Proceedings of ATAED 2018, CEUR Workshop Proceedings 2115, pp. 91–111. 2018

[2] Y. Dimopoulos, E. Kouppari, A. Philippou, and K. Psara. *Encoding Reversing Petri Nets in Answer Set Programming*. In Proceedings of the 12th International Conference on Reversible Computation (pp. 264-271). Lecture Notes in Computer Science volume 12227. Springer 2020

Thank you! Questions?