Faculty of Computer Science

ACM Sigmod

Moscow
2025

Process Models with Data: Soundness Verification and Repair

Student:
Nikolai M. Suvorov,
Research Assistant of PAIS Lab,
HSE University

Scientific Advisor:
Irina A. Lomazova,
D.Sc., Professor, Head of the PAIS Lab

Processes and Process Models

A business process is a distributed process consisting of a set of activities that are performed in coordination in an organizational and technical environment and allow achieving a business goal.

Analysis of such processes is usually done on their models.

Analysis helps to find process inconsistencies and vulnerabilities and provides information that may form a basis for making decisions to improve and optimize processes.

M. Weske, Introduction, Springer Berlin Heidelberg, Berlin, Heidelberg,2012, pp. 3–23.

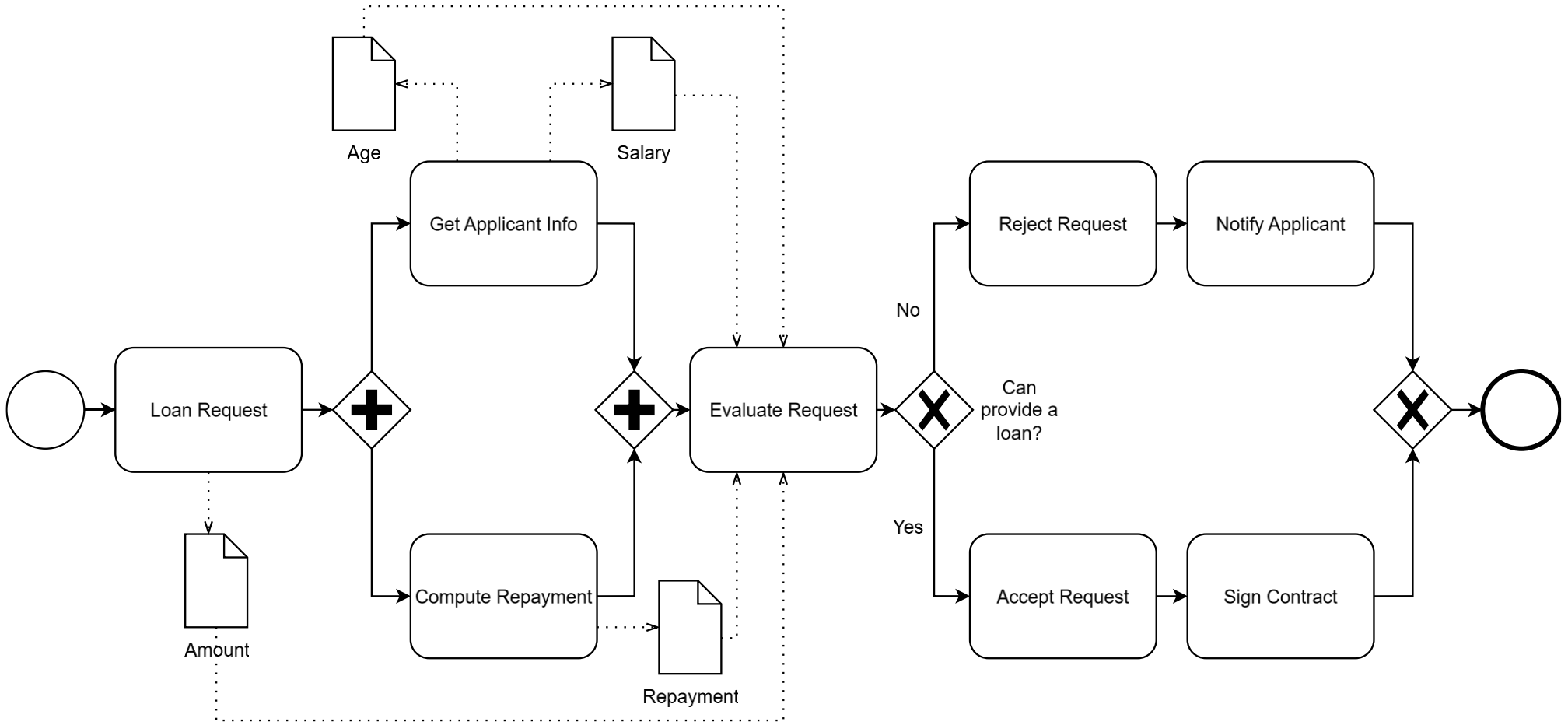
Process Model Soundness

A process model is sound if:

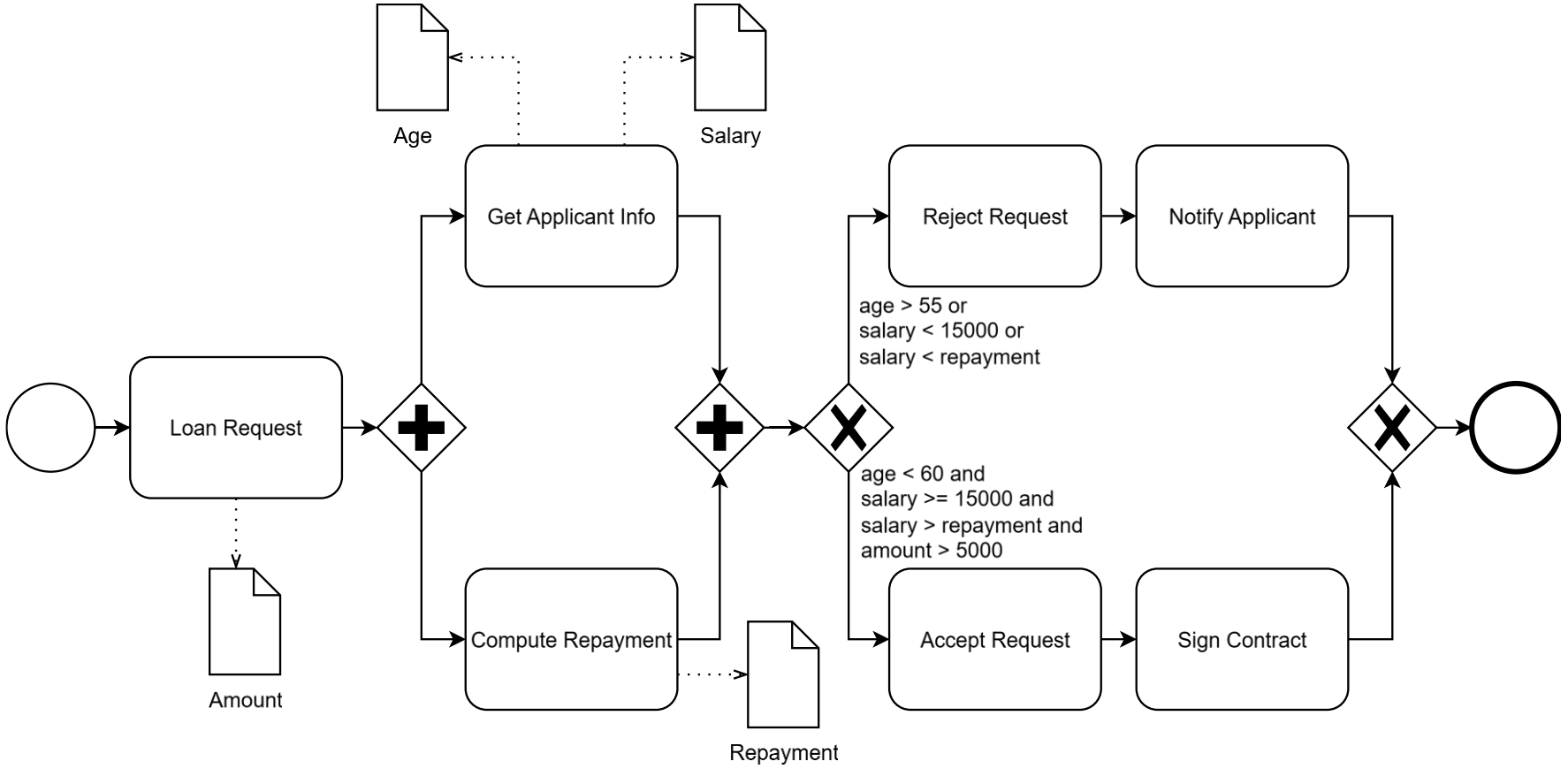
- The process always properly terminates.
- Each process model activity can occur in a process instance.

How to verify soundness of *data-aware* processes?

Loan Application Process

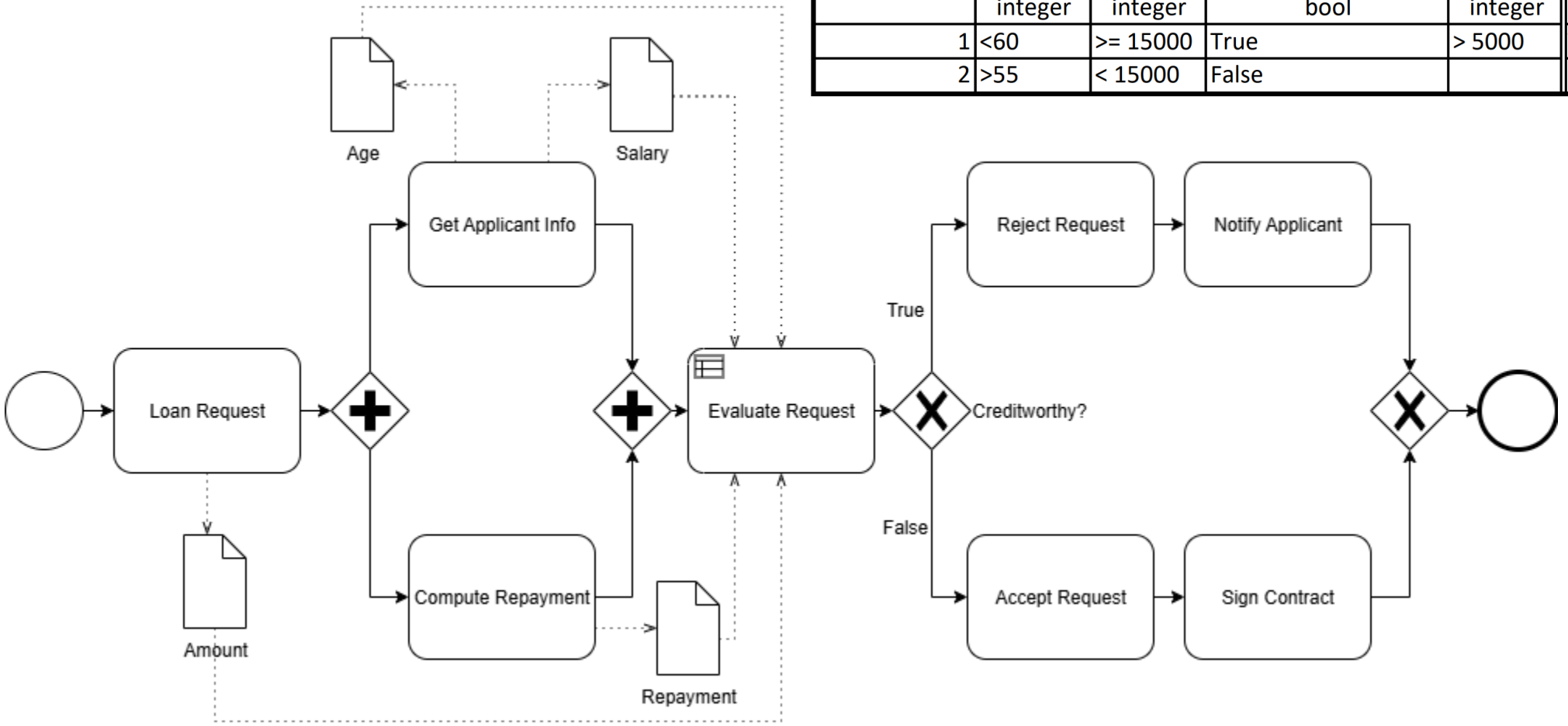


BPMN Extension with conditions on arcs



BPMN with DMN tables

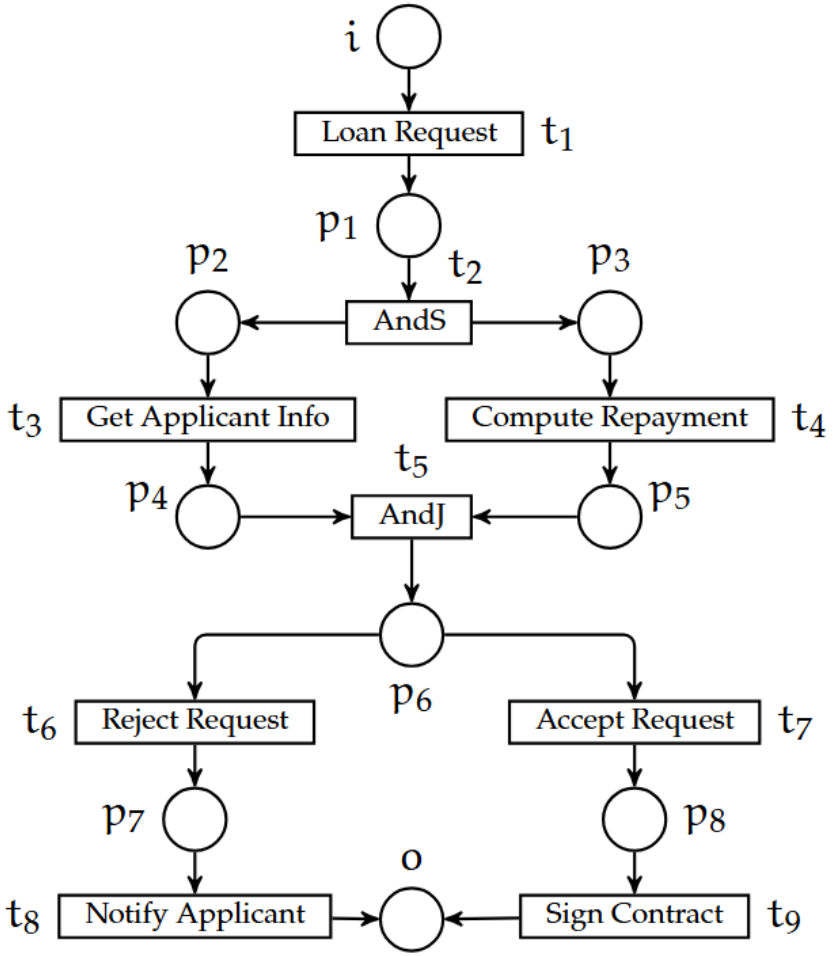
Creditworthy Decision					
U	Age	Salary	{Repayment < Salary}	Amount	Creditworthy
	integer	integer	bool	integer	bool
1	<60	>= 15000	True	> 5000	True
2	>55	< 15000	False		False



Workflow nets

A WF-net is a place-transition system (P, T, F, l) with distinguished input (i) and output (o) places, where:

- P is a set of places,
- T is a set of transitions,
- F is a flow relation $(P \times T) \cup (T \times P) \rightarrow \mathbb{N}$,
- l is a labeling function $T \rightarrow A$,
- Every node from $(P \cup T)$ is on a path from i to o.
- i is a single source place, o is a single sink place.



W. M. van der Aalst, "The application of petri nets to workflow management", J. Circuits Syst. Comput., vol. 8, pp. 21–66, 1998.

Workflow nets with variables as places

We need to create a number of places to represent variables.

Three viable approaches:

1. A single place for **each possible variable value** (only for finite domains).
2. Two places **for each variable** (only for finite domains).
3. A single place for **each formula representing possible values** of variables. Tackle concurrency issues.

After that, add copies of transitions that check or update variable values.

Workflow nets with variables as places

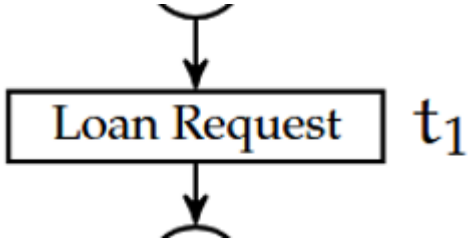
We need to create a number of places to represent variables.

Three viable approaches:

1. A single place for **each possible variable value** (only for finite domains).
2. Two places **for each variable** (only for finite domains).
3. A single place for **each formula representing possible values** of variables. Tackle concurrency issues.

After that, add copies of transitions that check or update variable values.

Approach 1



Loan Request updates amount, amount is in range [1, 1 000 000]

Adding 999 999 LR transitions
Adding 1 000 000 places

Workflow nets with variables as places

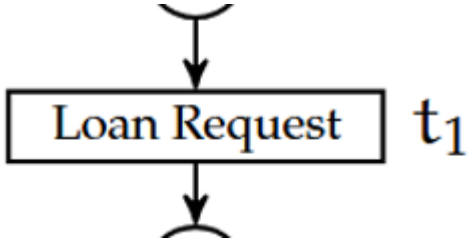
We need to create a number of places to represent variables.

Three viable approaches:

1. A single place for **each possible variable value** (only for finite domains).
2. Two places **for each variable** (only for finite domains).
3. A single place for **each formula representing possible values** of variables. Tackle concurrency issues.

After that, add copies of transitions that check or update variable values.

Approach 2



Loan Request updates amount, amount is in range [1, 1 000 000]

Adding two places:
one without tokens,
one with 1 000 000 tokens.
Adding 999 999 transitions.

Workflow nets with variables as places

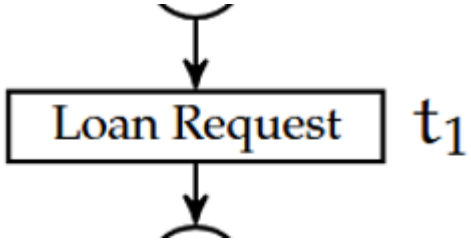
We need to create a number of places to represent variables.

Three viable approaches:

1. A single place for **each possible variable value** (only for finite domains).
2. Two places **for each variable** (only for finite domains).
3. A single place for **each formula representing possible values** of variables. Tackle concurrency issues.

After that, add copies of transitions that check or update variable values.

Approach 3



Loan Request updates amount, amount is compared with 5000

Adding 2 LR transitions and 3 places
(amount < 5000)
(amount = 5000)
(amount > 5000)

Workflow nets with variables as places

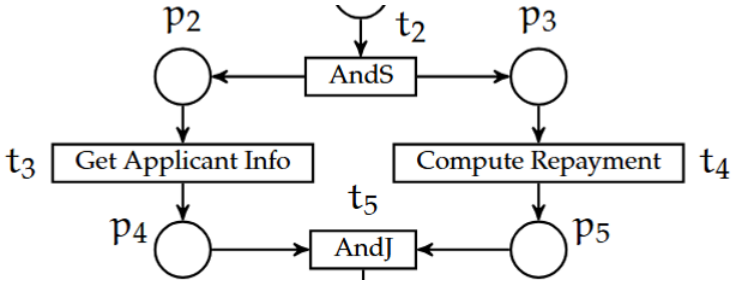
We need to create a number of places to represent variables.

Three viable approaches:

1. A single place for **each possible variable value** (only for finite domains).
2. Two places **for each variable** (only for finite domains).
3. A single place for **each formula representing possible values** of variables. Tackle concurrency issues.

After that, add copies of transitions that check or update variable values.

Approach 3



GAI updates age & salary,
CR updates repayment,
age is compared with 55,60,
salary is compared with 15000 & repayment

5 places for age (<55,55,(55,60),60,>60)
19 places for salary & repayment
(s < 15000 & r undef)
(s < 15000 & r < salary)
(s < 15000 & r = salary)
(s < 15000 & r > s & r < 15000)
(s < 15000 & r > 15000)
(s < 15000 & r = 15000)
...

Workflow nets with variables as places

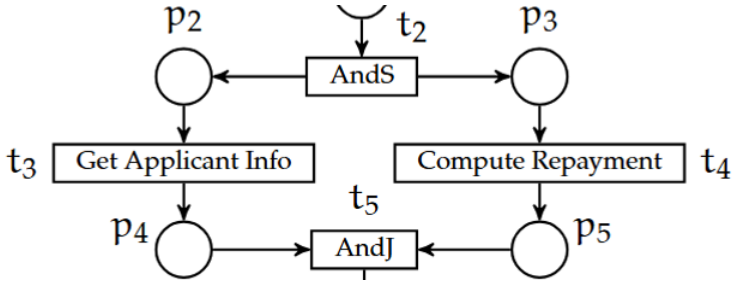
We need to create a number of places to represent variables.

Three viable approaches:

1. A single place for **each possible variable value** (only for finite domains).
2. Two places **for each variable** (only for finite domains).
3. A single place for **each formula representing possible values** of variables. Tackle concurrency issues.

After that, add copies of transitions that check or update variable values.

Approach 3



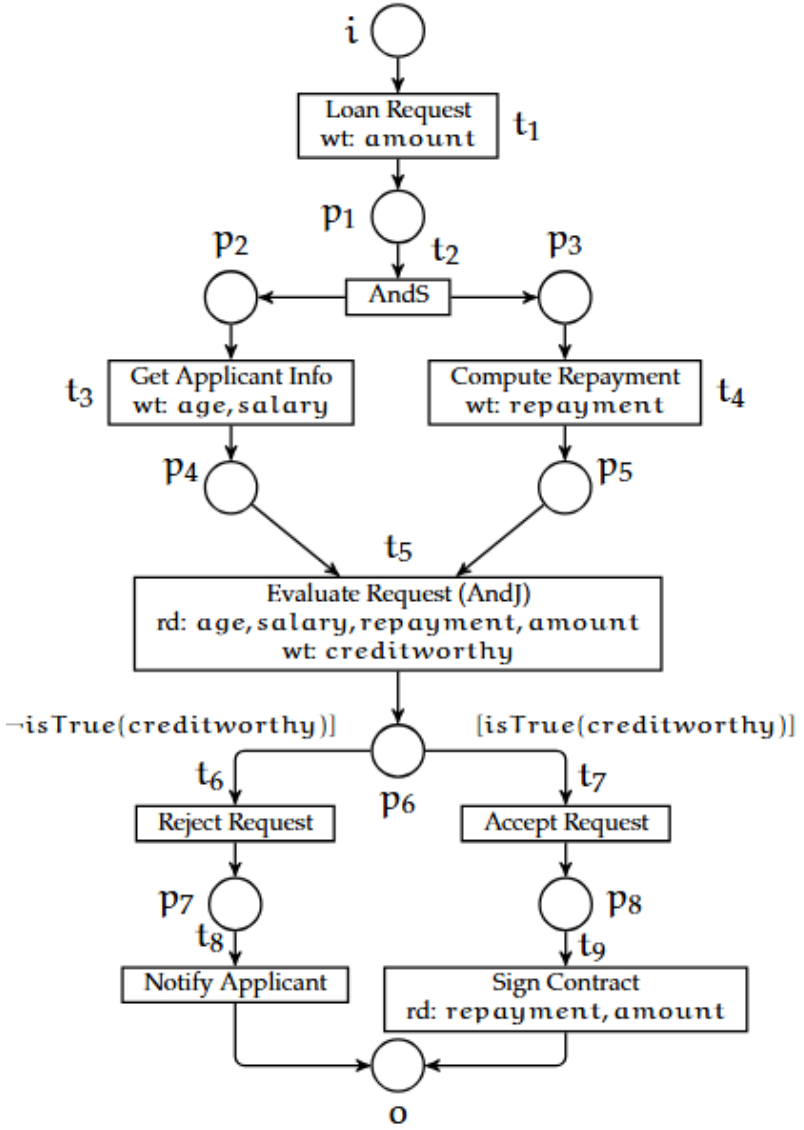
GAI updates age & salary,
CR updates repayment,
age is compared with 55,60,
salary is compared with 15000 & repayment

Can make write on AndJ.
Add 13x5 AndJ transitions and 18 places.
5 places for age,
13 places for s & r (without undef)

Workflow nets with data

WDF-net $N = (P, T, F, l, rd, wt, del, grd)$ is a WF-net that may read, write and delete the **data elements** on transition firings.
Transitions may have **guards** that are predicates on the data elements and that can block execution.

A **state** in a WFD-nets includes the marking, the state of data elements (def or undef), and the state of guards (T,F, or undef).

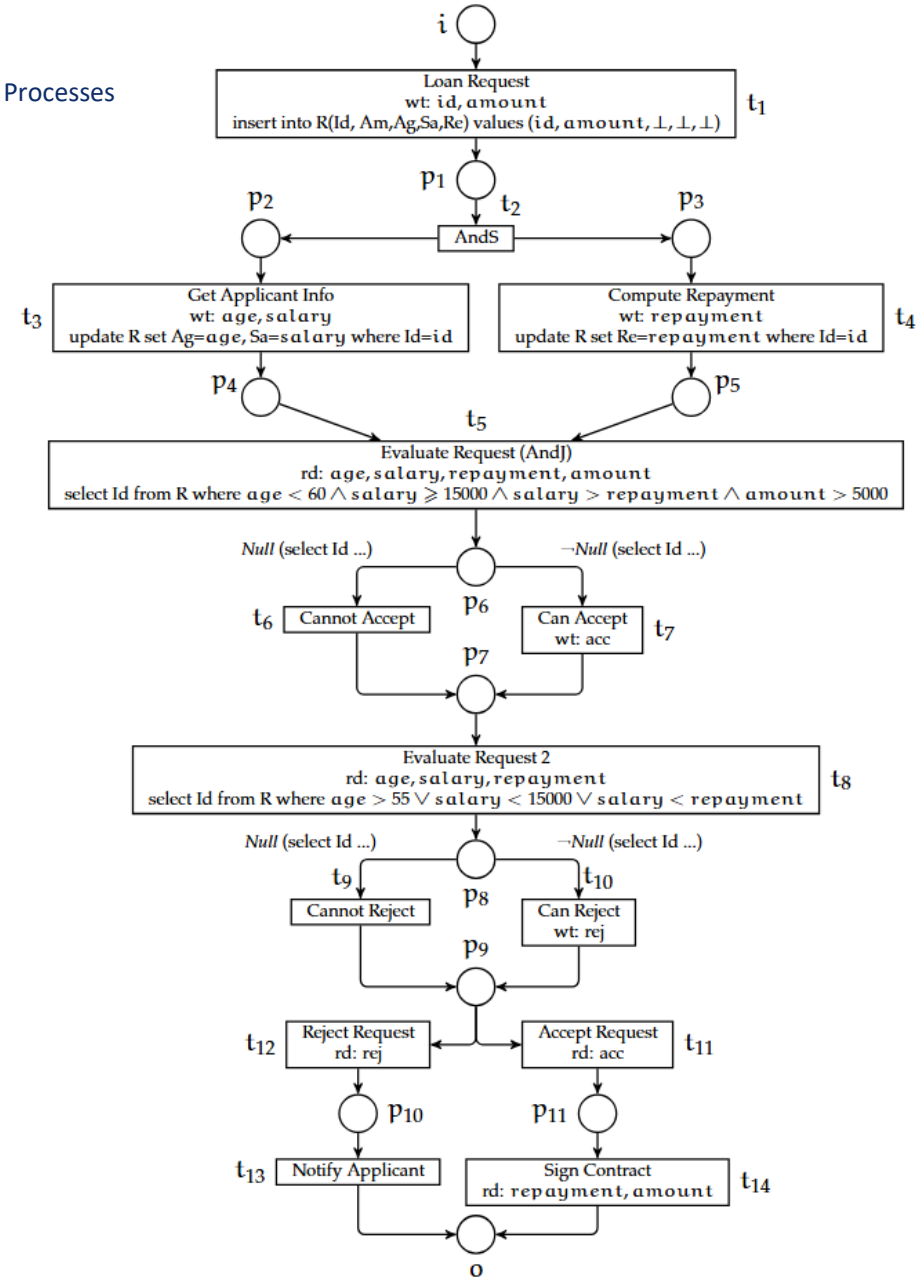


N. Sidorova, C. Stahl, and N. Trčka, "Soundness verification for conceptual workflow nets with data: Early detection of errors with the most precision possible", Inf. Syst., vol. 36, no. 7, pp. 1026–1043, Nov. 2011.

Workflow nets with tables

WFT-net $N = (P, T, F, l, rd, wt, del, grd, R, O_R)$ is a WFD-net with relational tables. Transitions may execute Select, Insert, Update, Delete queries. Results of these queries may be used in guards.

A **state** in a WFT-nets includes the marking, the state of data elements (def or undef), the state of guards (T,F, or undef), and the state of tables.



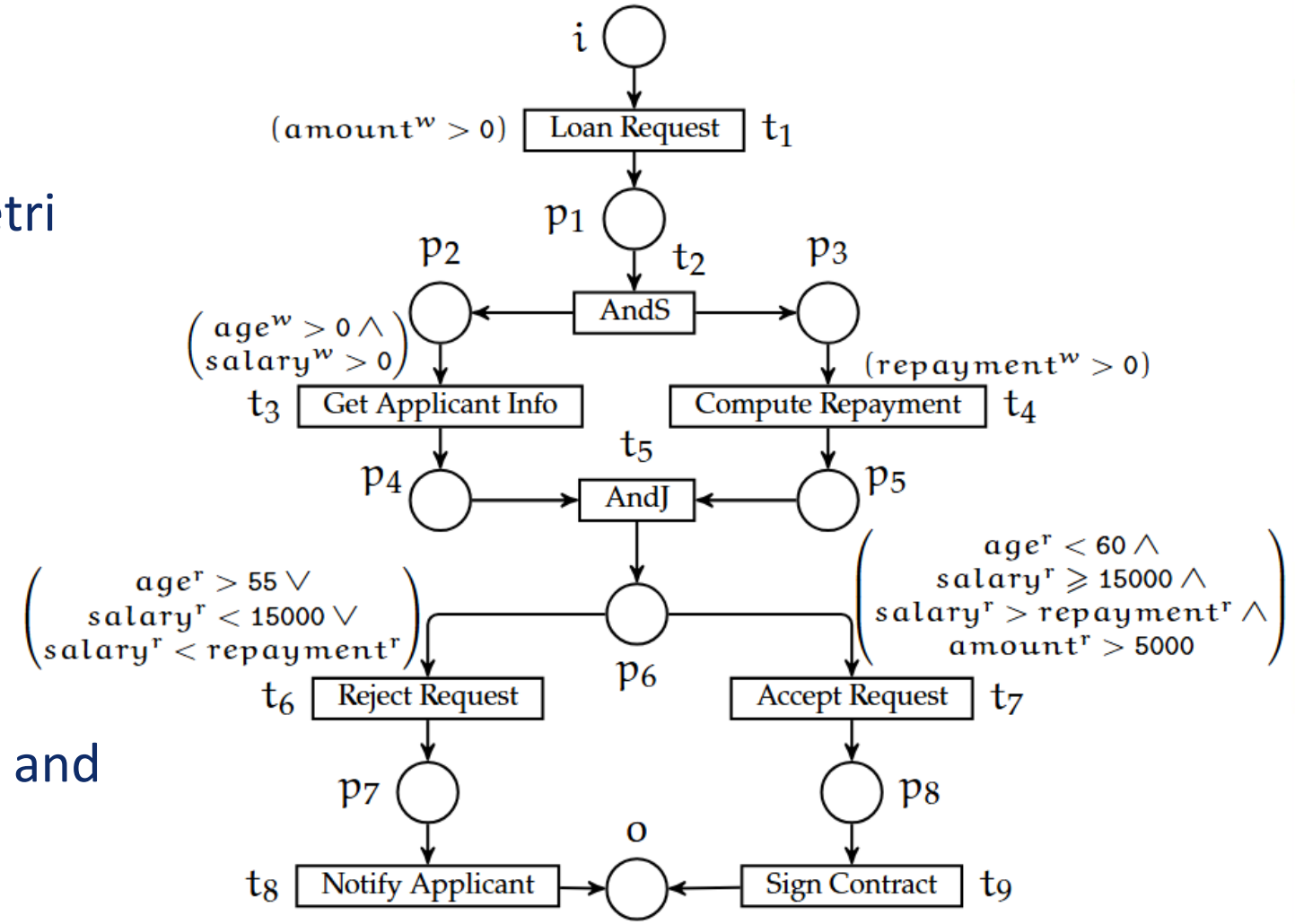
X. Tao, G. Liu, B. Yang, C. Yan, and C. Jiang, “Workflow nets with tables andtheir soundness”, IEEE Transactions on Industrial Informatics, vol. 16, no. 3,pp. 1503–1515, 2020.

Data Petri nets

DPN $N = (P, T, F, l, V, guard)$ is a Petri net, where:

- V is a finite set of variables,
- guard is a guard labeling function

$$T \rightarrow \Phi(V^r \cup V^w)$$



A **state** in a DPNs includes the marking and the current variable values.

M. de Leoni, J. Munoz-Gama, J. Carmona, and W. M. P. van der Aalst, "Decomposing alignment-based conformance checking of data-aware process models", in On the Move to Meaningful Internet Systems: OTM 2014 Conferences, R. Meersman, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 3–20.

Model Failure Points

Marking:

$M = [p_6]$

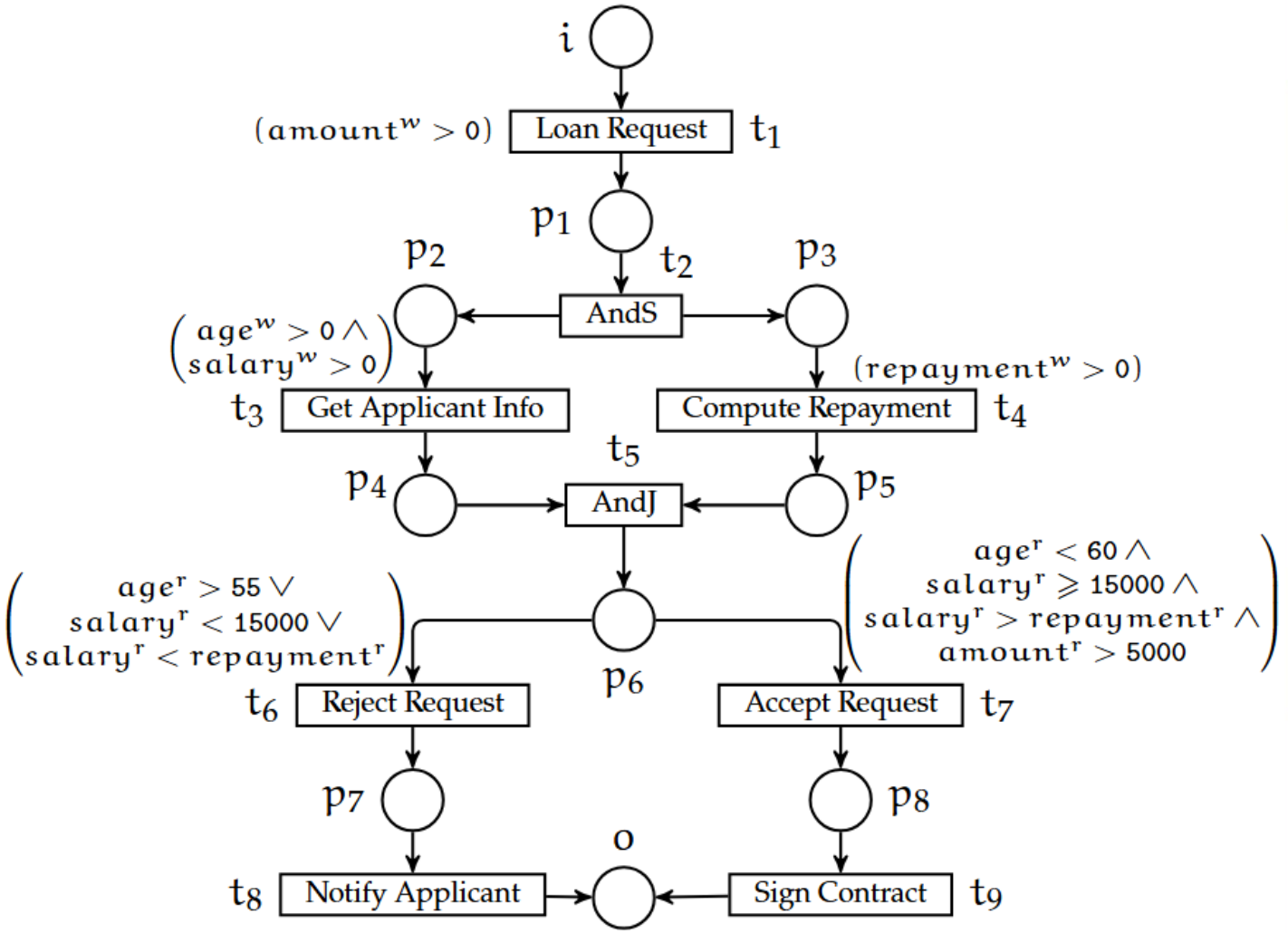
Variable state:

$\alpha(age) = 50$

$\alpha(salary) = 16000$

$\alpha(repayment) = 1000$

$\alpha(amount) = 4500$



What to Use

WF-nets: small number of variables with small domains (bool, enum, string).

WFD-nets: important existence of resources at the moment of an activity, very simple guards that do not relate on each other.

WFT(C)-nets: same for WFD-nets, where a process operates with data stores.

DPNs: data can be only read and written, comparison operators between constants and variables, variables of both finite/infinite domains, variables may be interrelated.

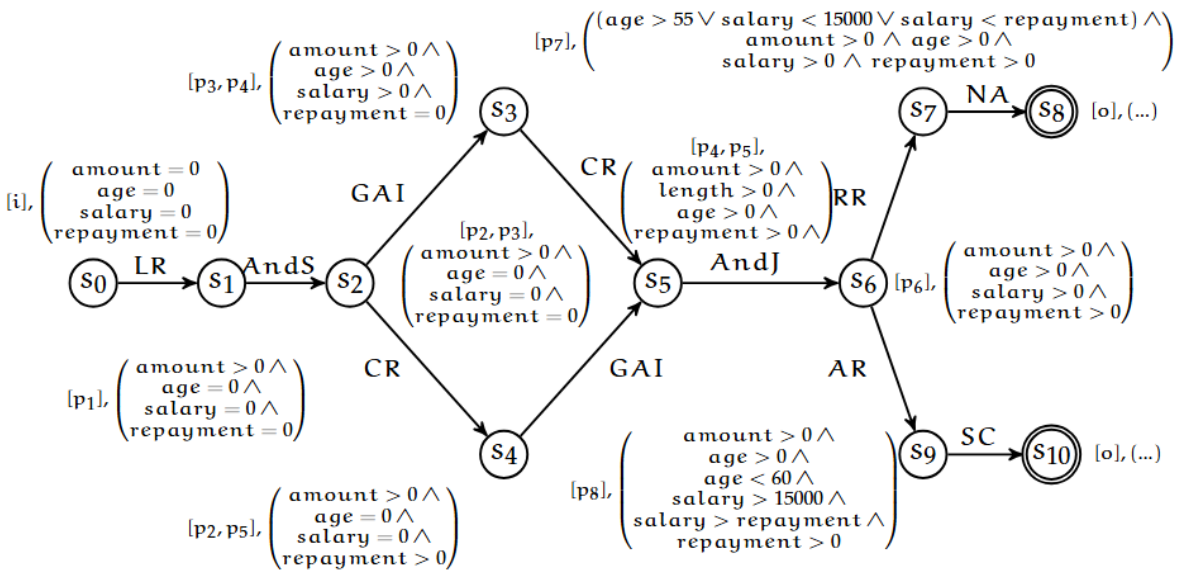
Exploring State Space for Soundness

WF-net:
- Coverability graph is finite.
- Reachability graph is finite if the net is bounded.

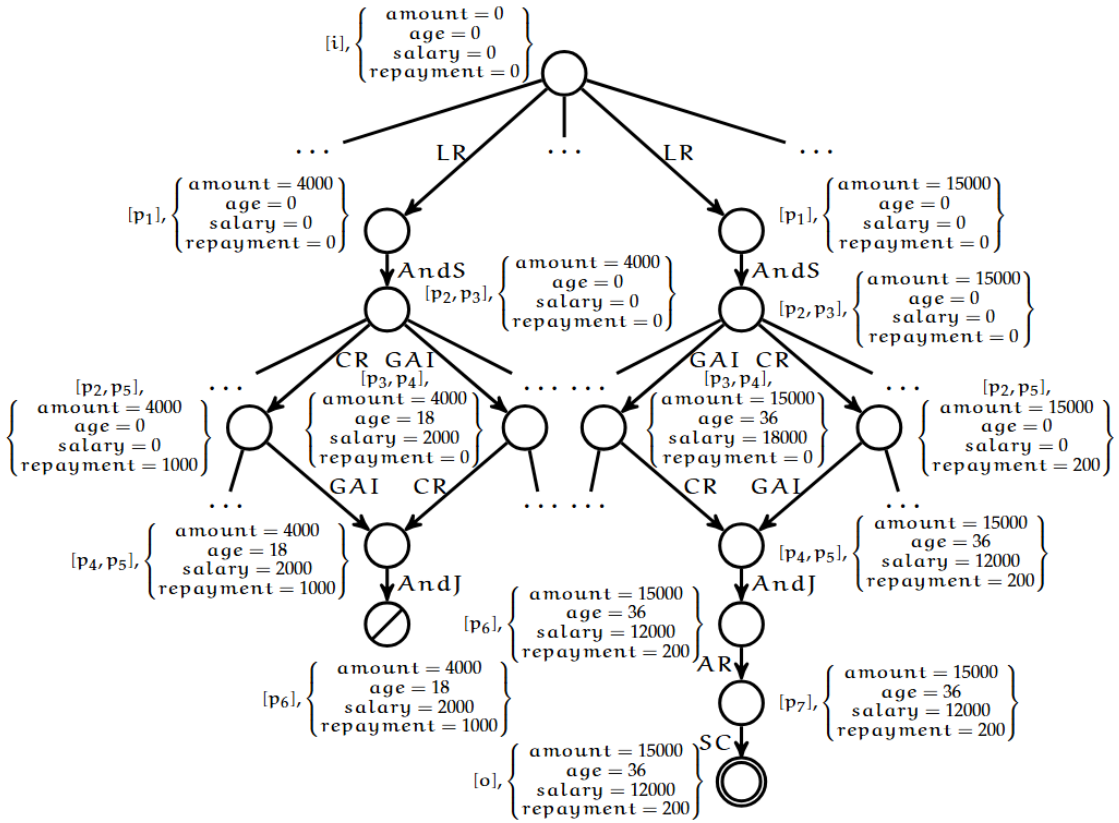
WFD-net, WFT-net, WFTC-net:
- Boundedness is assumed.
- Reachability graph is finite if the net is bounded.

DPN:
- Coverability graph may be not finite.
- Reachability graph may be not finite even if the net is bounded.

DPN State Space Structures



Abstract Reachability Graph



Reachability Graph

Some properties of ACG/ARG

Properties that can be checked on ACG/ARG:

- Boundedness
- Absence of dead transitions
- Reachability of [o] if the net is bounded, reachability of $M \geq [o]$
- Absence of deadlocks & livelocks on the backbone level (Petri net) if the net is bounded.

Properties that cannot be checked on ACG/ARG:

- Absence of deadlocks & livelocks caused by adding the dataflow.
- Reachability of [o] if the net is unbounded.

Soundness: Ways To Address

1. Split ARG nodes to make them as granular as possible w.r.t. variables updated on preceding transition firing -> **huge ARG**.
2. Construct an ARG for each reachable marking, unite the obtained results -> **huge number of ARGs**.

P. Felli, M. Montali, S. Winkler, Soundness of data-aware processes with arithmetic conditions, Advanced Information Systems Engineering, Springer, 2022, pp. 389–406.
P. Felli, M. de Leoni, M. Montali, Soundness verification of data-aware process models with variable-to-variable conditions, Fundam. Informaticae 182 (1) (2021) 1–29.

Soundness: Ways To Address

1. Split ARG nodes to make them as granular as possible w.r.t. variables updated on preceding transition firing -> **huge ARG**.
2. Construct an ARG for each reachable marking, unite the obtained results -> **huge number of ARGs**.
3. Refine DPN, so that an ARG of the refined DPN is sufficient for checking deadlocks and livelocks -> **slightly bigger ARG, several number of ARGs**.

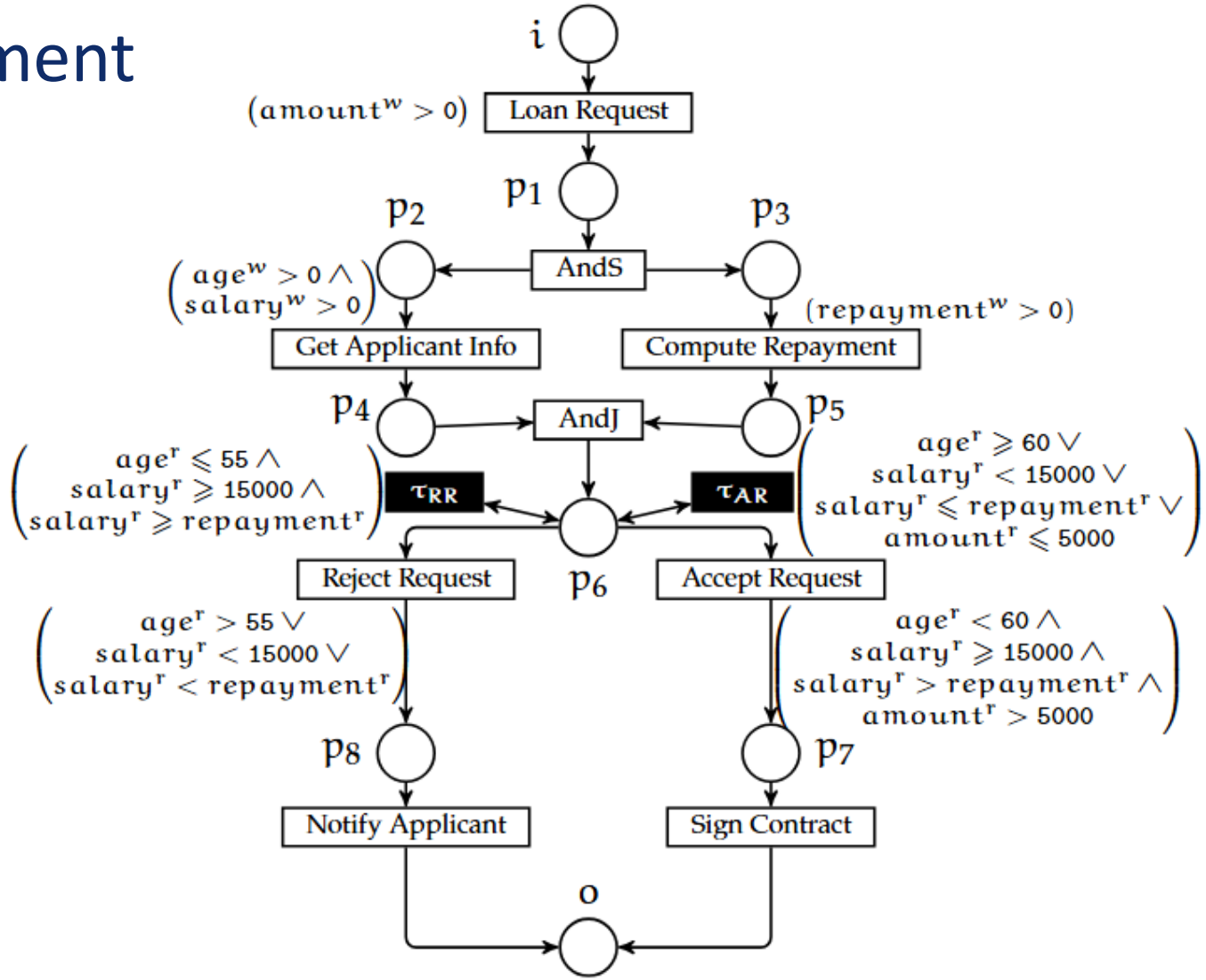
P. Felli, M. Montali, S. Winkler, Soundness of data-aware processes with arithmetic conditions, Advanced Information Systems Engineering, Springer, 2022, pp. 389–406.
P. Felli, M. de Leoni, M. Montali, Soundness verification of data-aware process models with variable-to-variable conditions, Fundam. Informaticae 182 (1) (2021) 1–29.

Deadlocks and DPN Refinement

Added two silent transitions with guards as negations of input conditions of Reject Request and Accept Request.

This allows to detect a deadlock on an ARG:

- Now an ARG includes a run with a consequent execution of τ_{RR} and τ_{AR}.
- Conjunction of their guards is a condition at which neither RR nor AR may fire.
- Thus, now an ARG contains a **dead node** which denotes a deadlock.

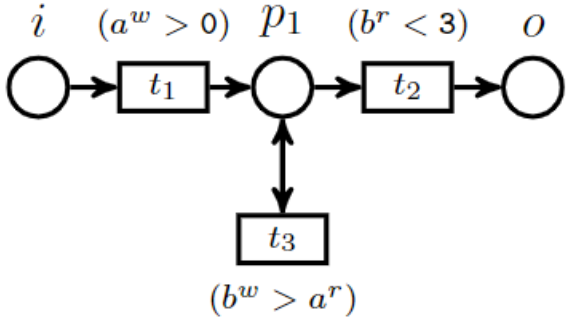


Livelocks and DPN Refinement

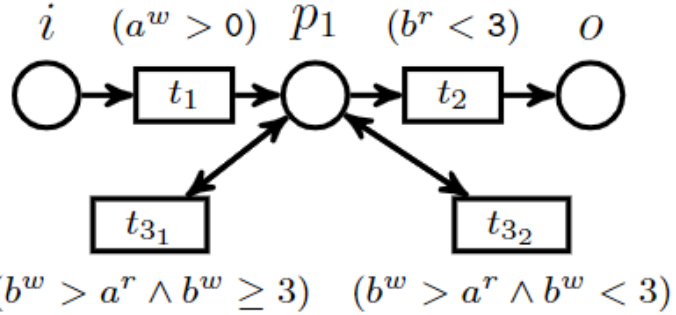
We may miss livelocks that (in a loop) overwrite variables checked in silent transitions.

Before adding silent transitions, we can split transitions from ARG loops based on the input condition of the transition leading out of a cycle.

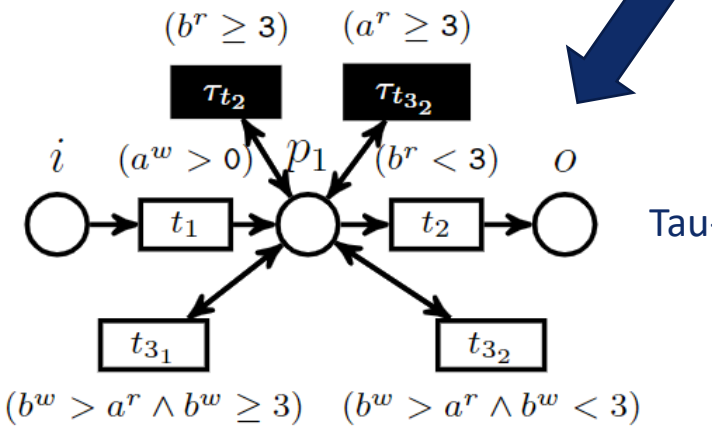
If there is a livelock, using this technique, we obtain a loop in the ARG, from which neither of output transitions may fire.



Source DPN



Refined DPN



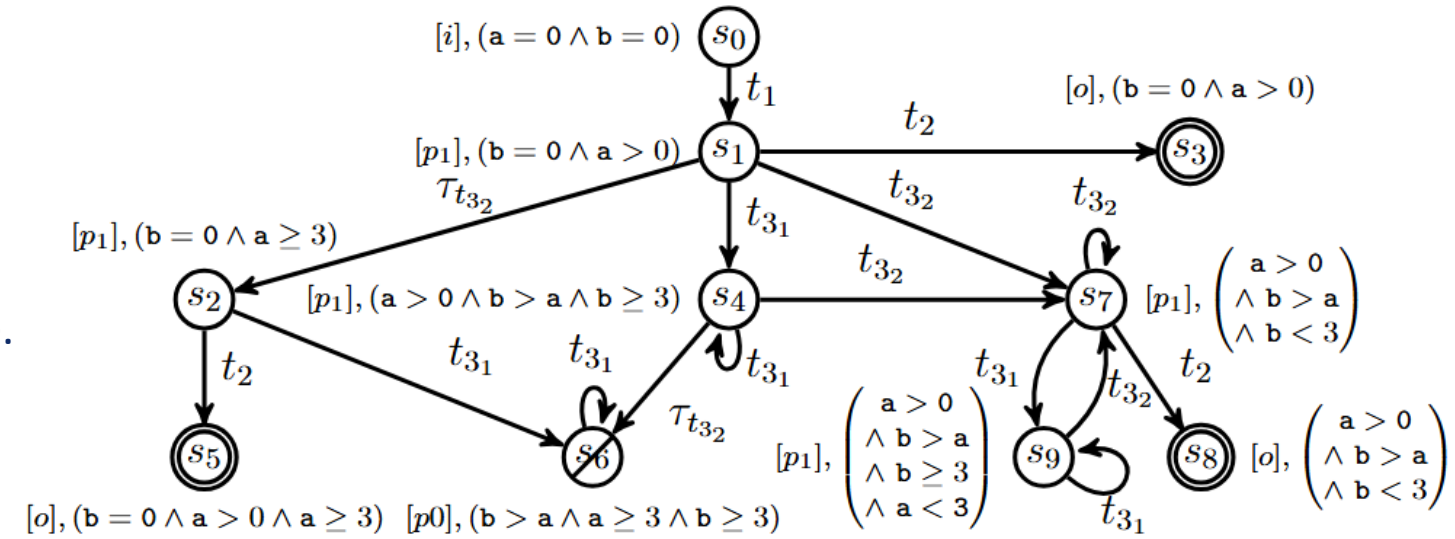
Tau-Refined DPN

Livelocks and DPN Refinement

We may miss livelocks that (in a loop) overwrite variables checked in silent transitions.

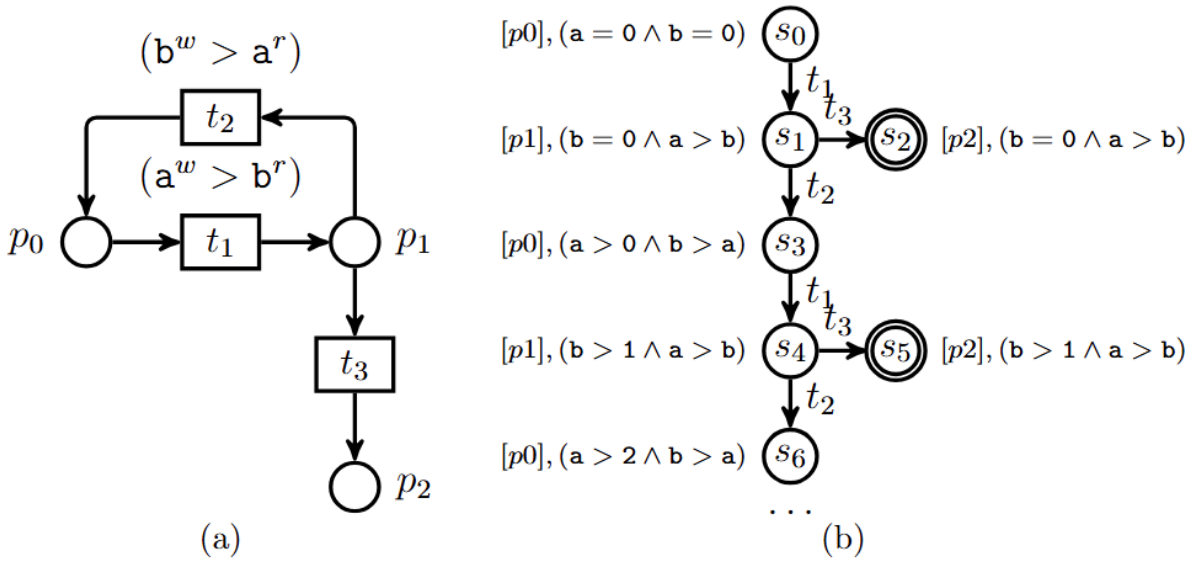
Before adding silent transitions, we can split transitions from ARG loops based on the input condition of the transition leading out of a cycle.

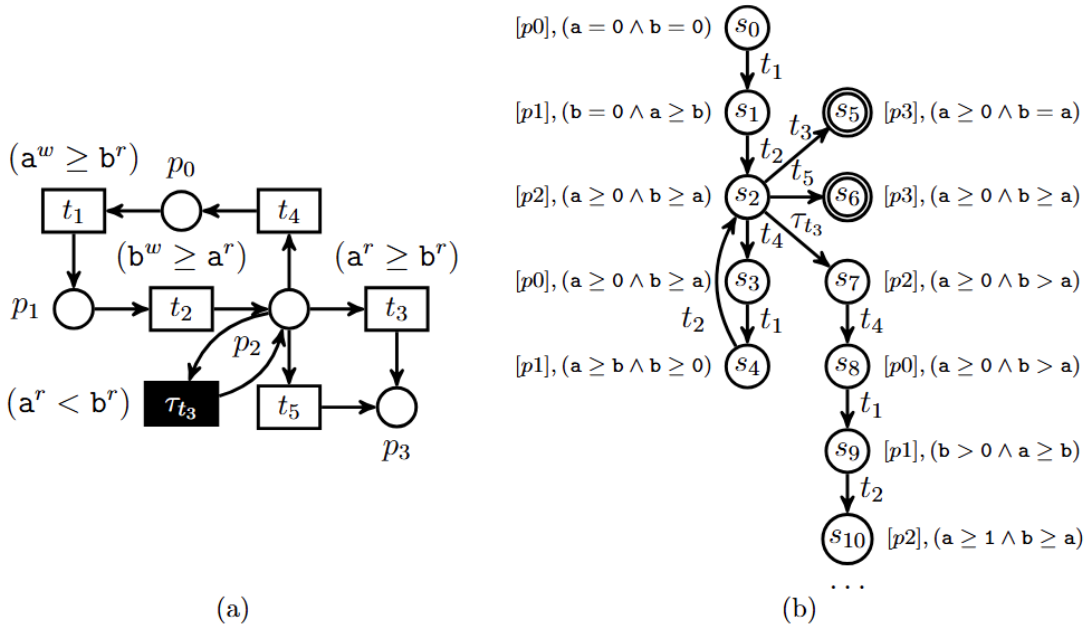
If there is a livelock, using this technique, we obtain a loop in the ARG, from which neither of output transitions may fire.



ARG

Applicability Borders

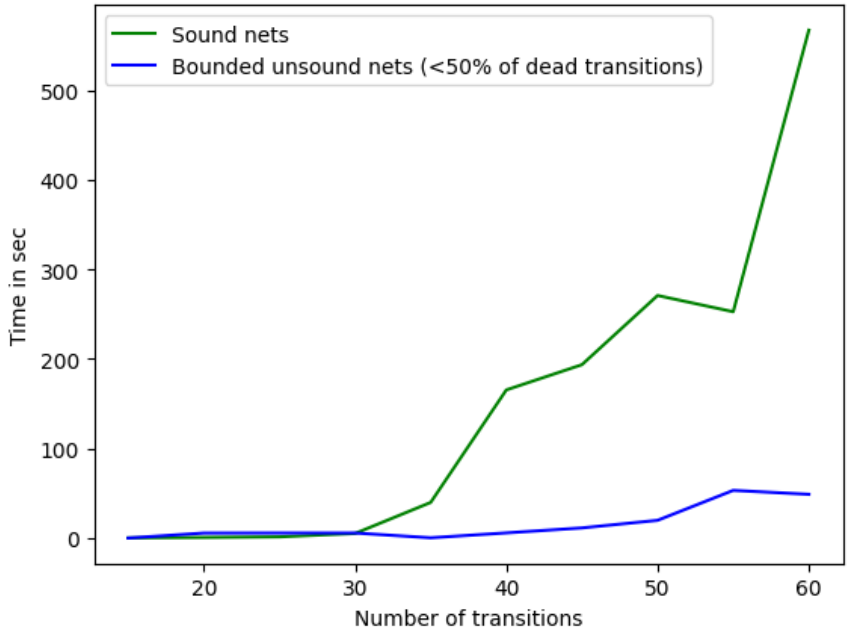




Verification Time

Given $n \in \mathbb{N}$, we consider DPNs of the following setup:

- $1.2n$ places,
- n transitions,
- $2.15n$ arcs for unsound DPNs and $2n$ arcs for sound DPNs,
- $0.25n$ variables, and
- $0.5n$ conditions.



Intel Core I7-7700HQ

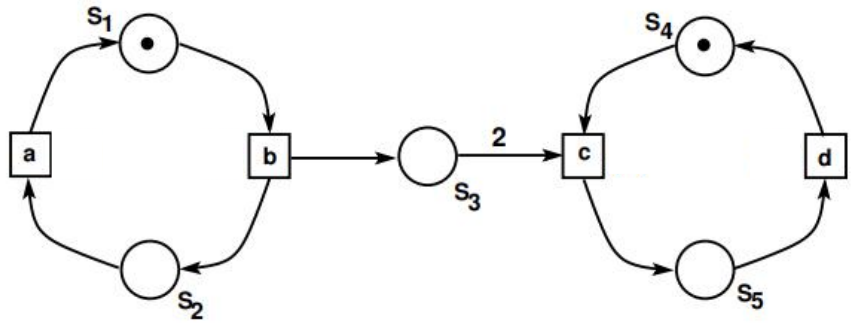
On the literature examples, nearly 1.5 times faster than the algorithm based on ARG construction for each marking

Soundness Repair Approaches

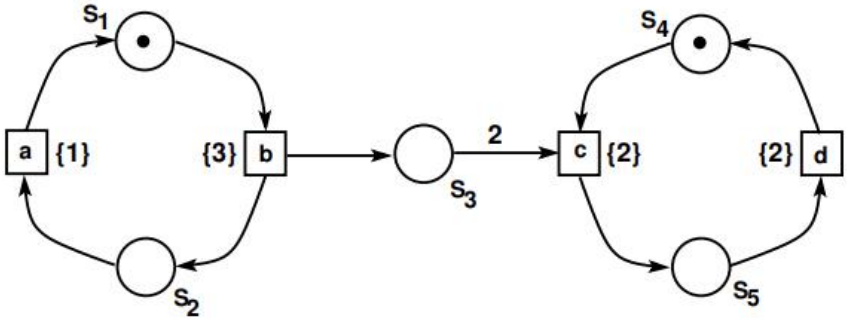
1. Restrict the allowed behavior.
2. Allow new behavior that previously led to failure points.
3. Use heuristics algorithms making small model perturbations w.r.t. costs of actions.

Soundness Repair Approaches

1. **Restrict the allowed behavior.**
2. Allow new behavior that previously led to failure points.
3. Use heuristics algorithms making small model perturbations w.r.t. costs of actions.

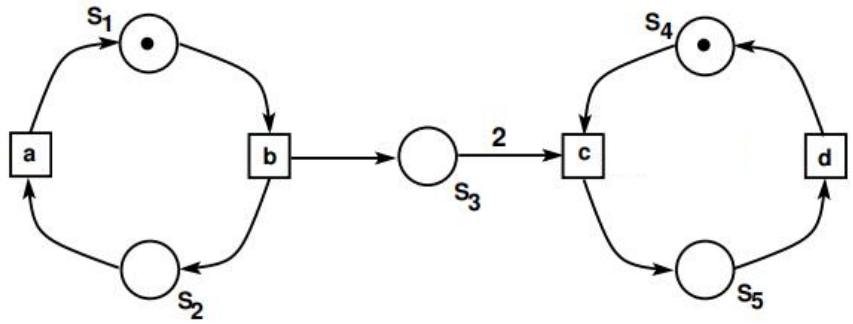


Adding Priorities to Transitions

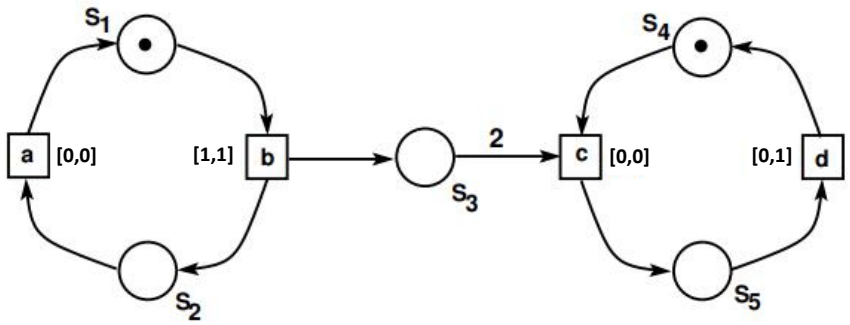


Soundness Repair Approaches

1. **Restrict the allowed behavior.**
2. Allow new behavior that previously led to failure points.
3. Use heuristics algorithms making small model perturbations w.r.t. costs of actions.

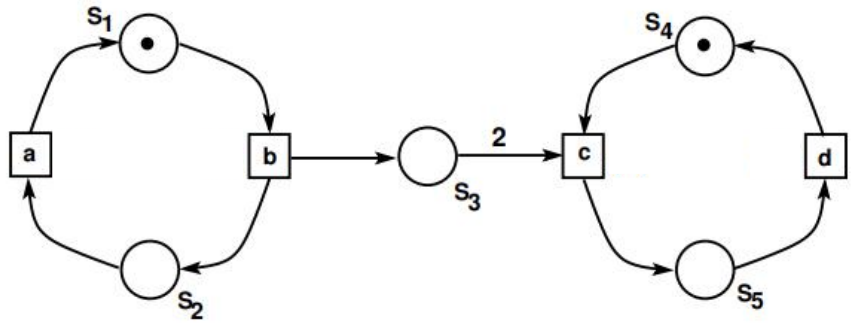


Adding Time Restrictions to Transitions

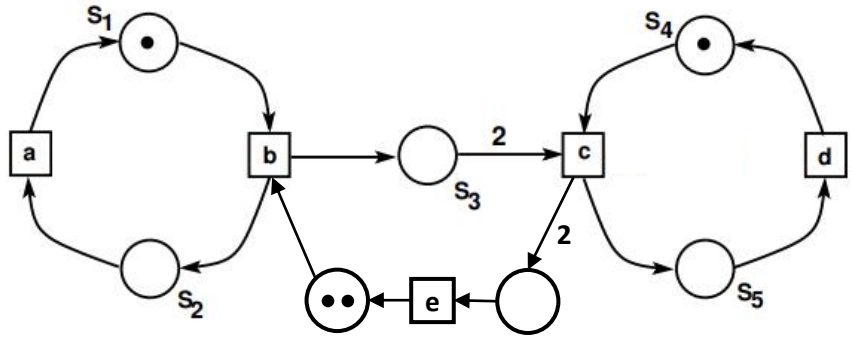


Soundness Repair Approaches

1. **Restrict the allowed behavior.**
2. Allow new behavior that previously led to failure points.
3. Use heuristics algorithms making small model perturbations w.r.t. costs of actions.

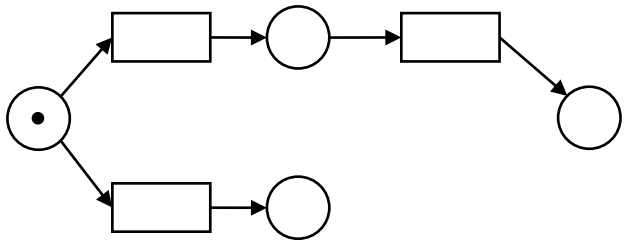


Adding Synthesized Controllers

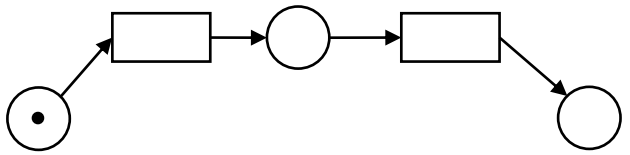


Soundness Repair Approaches

1. **Restrict the allowed behavior.**
2. Allow new behavior that previously led to failure points.
3. Use heuristics algorithms making small model perturbations w.r.t. costs of actions.

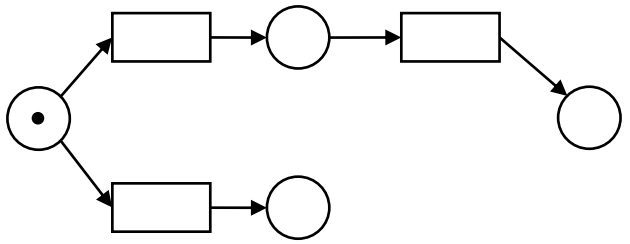


Removing PN unfeasible paths

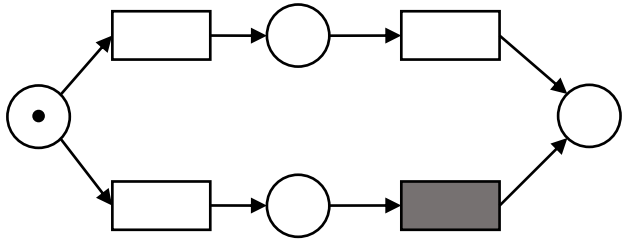


Soundness Repair Approaches

1. Restrict the allowed behavior.
2. **Allow new behavior that previously led to failure points.**
3. Use heuristics algorithms making small model perturbations w.r.t. costs of actions.

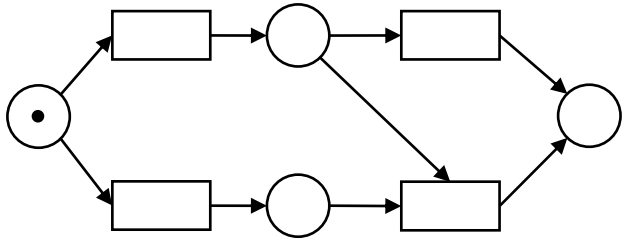


Adding extra transitions

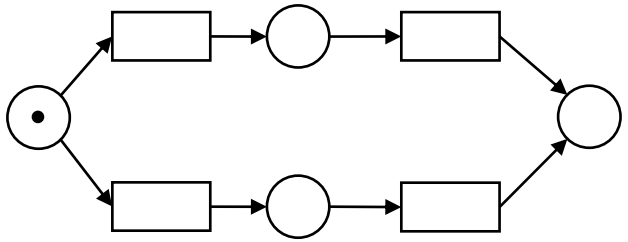


Soundness Repair Approaches

1. Restrict the allowed behavior.
2. **Allow new behavior that previously led to failure points.**
3. Use heuristics algorithms making small model perturbations w.r.t. costs of actions.

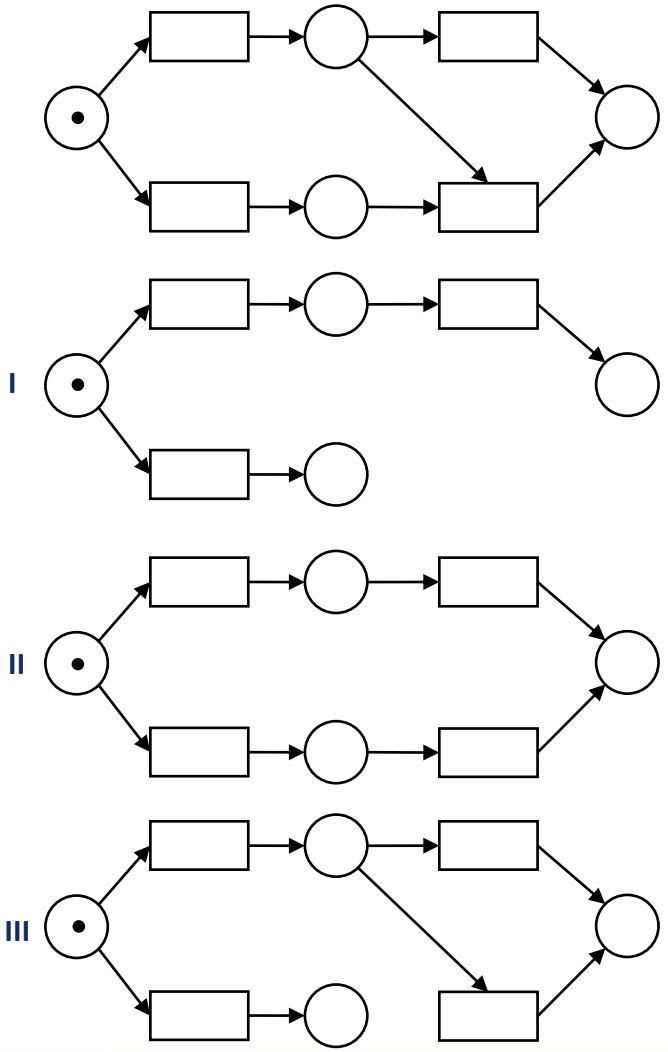


Removing extra arcs



Soundness Repair Approaches

1. Restrict the allowed behavior.
2. Allow new behavior that previously led to failure points.
3. **Use heuristics algorithms making small model perturbations w.r.t. costs of actions.**

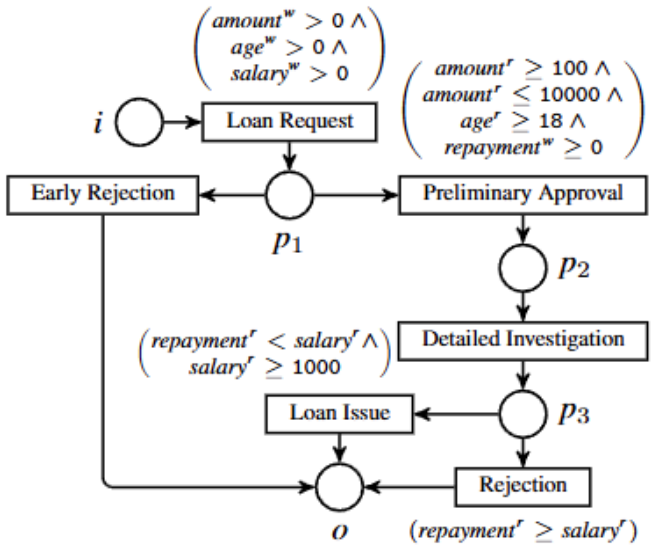


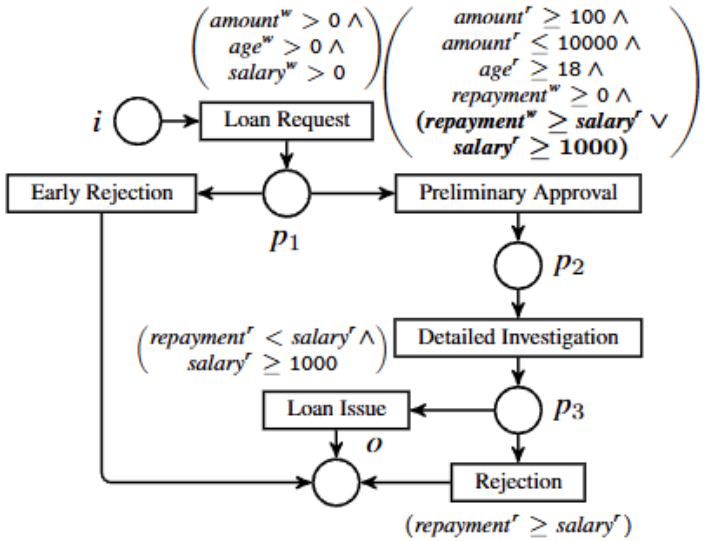
Soundness Repair of DPNs

1. Restrict the allowed behavior.
2. Allow new behavior that previously led to failure points.
3. Use heuristics algorithms making small model perturbations w.r.t. costs of actions.

Soundness Repair of DPNs

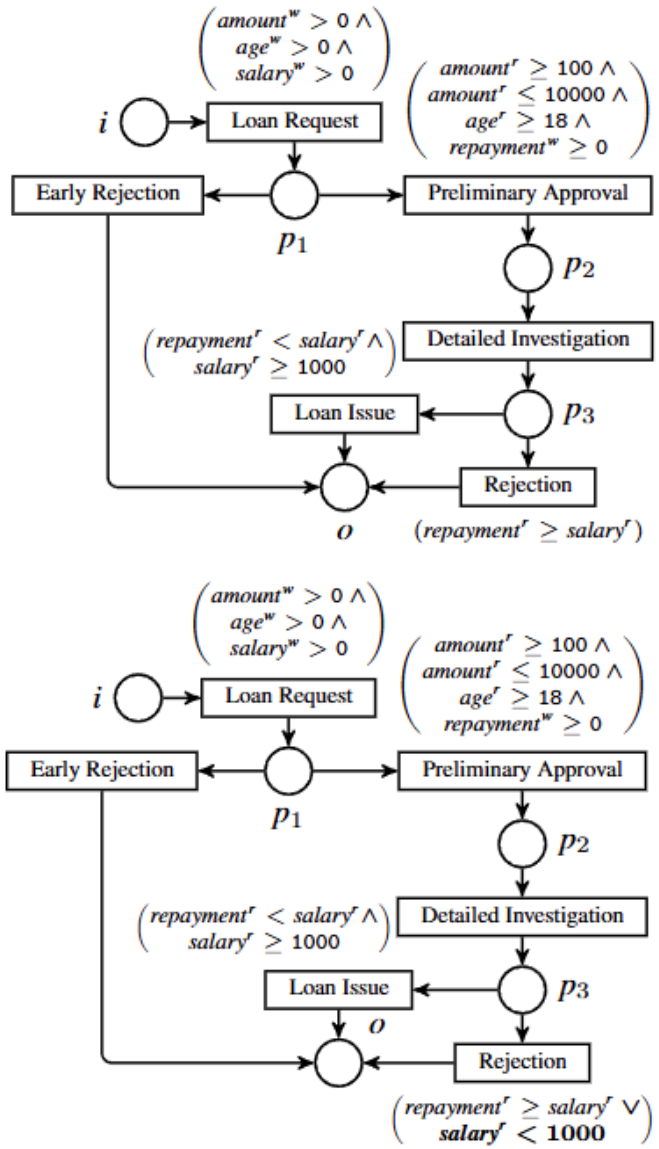
1. **Restrict the allowed behavior.**
2. Allow new behavior that previously led to failure points.
3. Use heuristics algorithms making small model perturbations w.r.t. costs of actions.



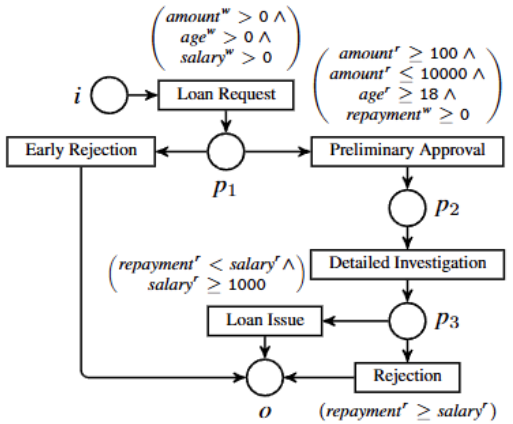


Soundness Repair of DPNs

1. Restrict the allowed behavior.
2. **Allow new behavior that previously led to failure points.**
3. Use heuristics algorithms making small model perturbations w.r.t. costs of actions.

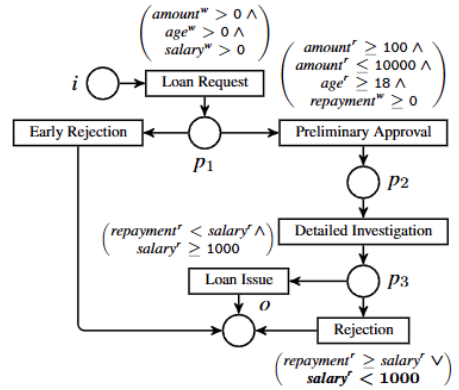


Soundness Repair of DPNs

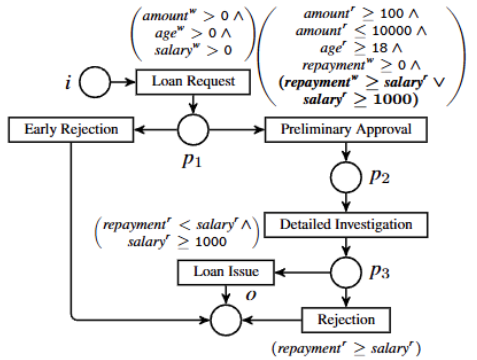


1. Restrict the allowed behavior.
2. Allow new behavior that previously led to failure points.
3. **Use heuristics algorithms making small model perturbations w.r.t. costs of actions.**

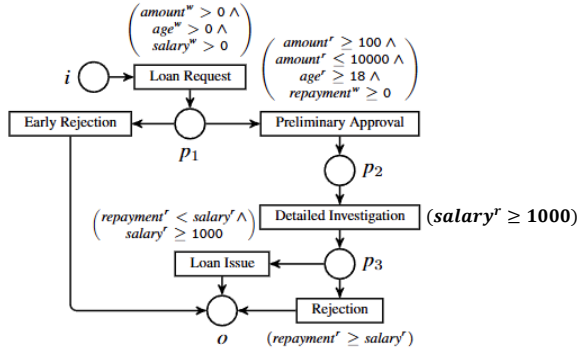
I



II



III

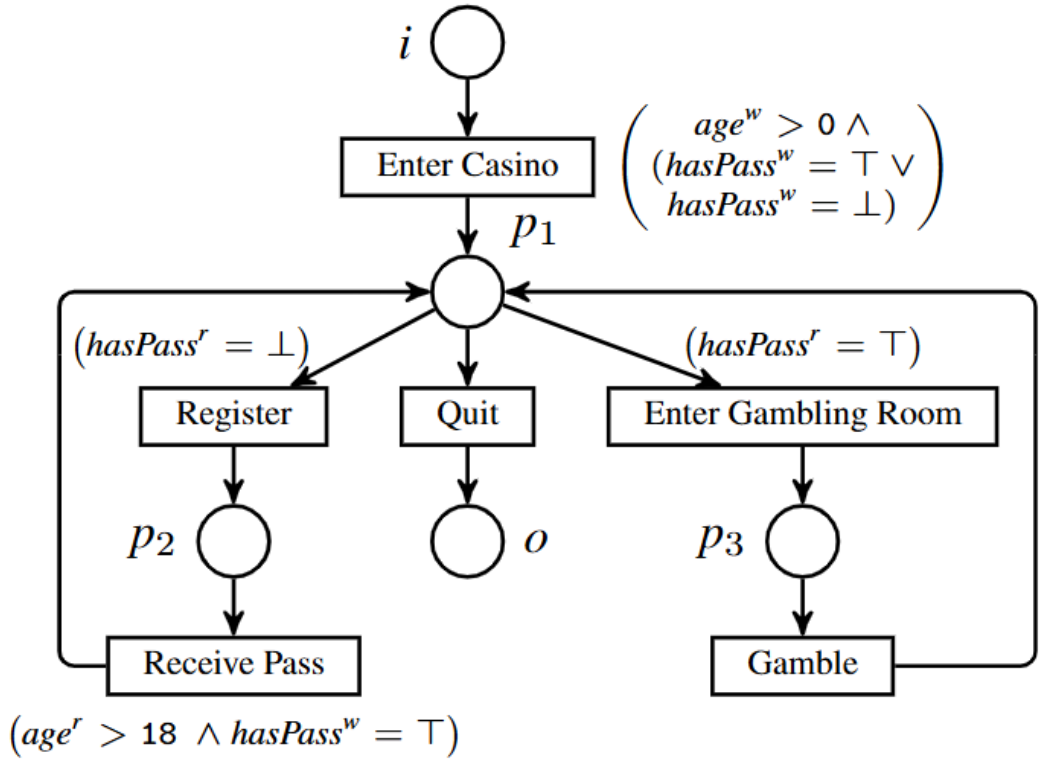


Soundness Repair of DPNs

Should the behavior not allowed in the model be allowed in the repaired model?

Soundness Repair of DPNs

Should the behavior not allowed in the model be allowed in the repaired model?
Probably, no.

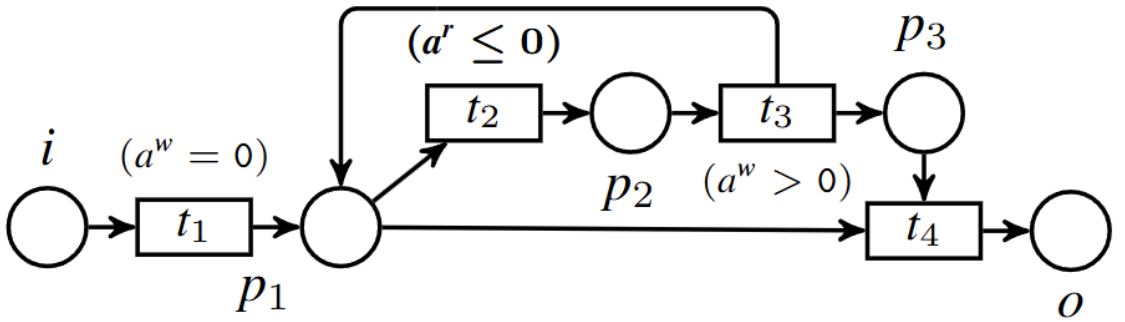


Soundness Repair of DPNs

Should soundness of the backbone be assumed?

Soundness Repair of DPNs

Should soundness of the backbone be assumed?
Probably, no.

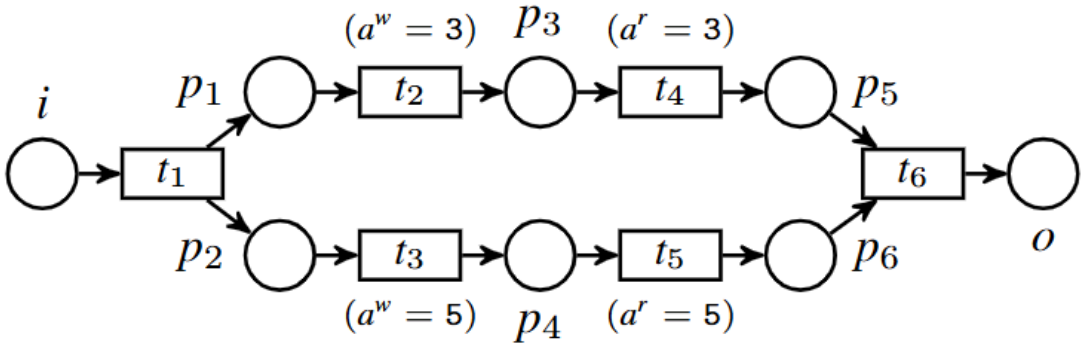


Sound DPN with an unsound backbone

Soundness Repair of DPNs

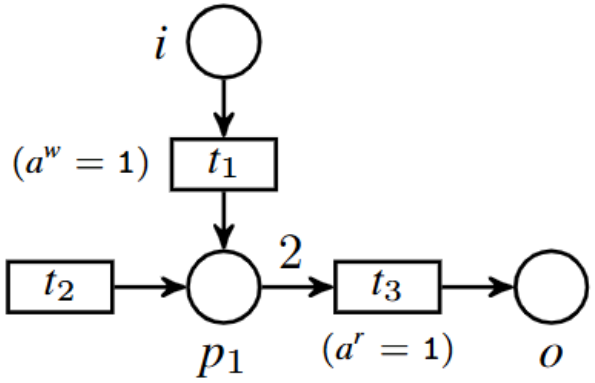
Can we propose a repair algorithm that is applicable in the general case and that always succeeds in repairing a model by restricting transition constraints?

Soundness Repair of DPNs



Irreparable DPN with a sound backbone

Can we propose a repair algorithm that is applicable in the general case and that always succeeds in repairing a model by restricting transition constraints?
No.



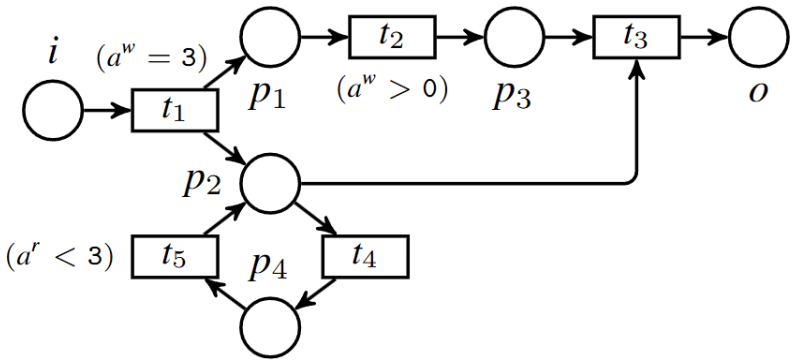
Irreparable DPN with an unsound backbone

Soundness Repair of DPNs

Can we propose a repair algorithm that if repairs, saves all the correct behavior of the source net in the resulting net obtained by restricting transition constraints?

Soundness Repair of DPNs

Can we propose a repair algorithm that if repairs, saves all the correct behavior of the source net in the resulting net obtained by restricting transition constraints?
No.



DPN for which it is impossible to save all the correct behavior after the repair using transition constraints restriction

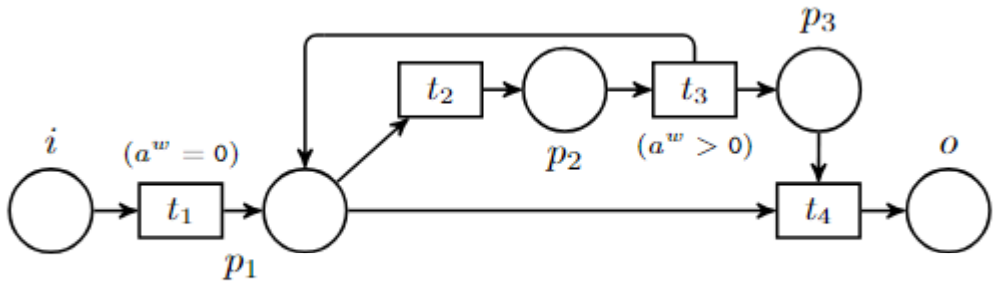
Soundness Repair of DPNs

But a semi-decision procedure for repairing DPNs could be developed

Repair Approach

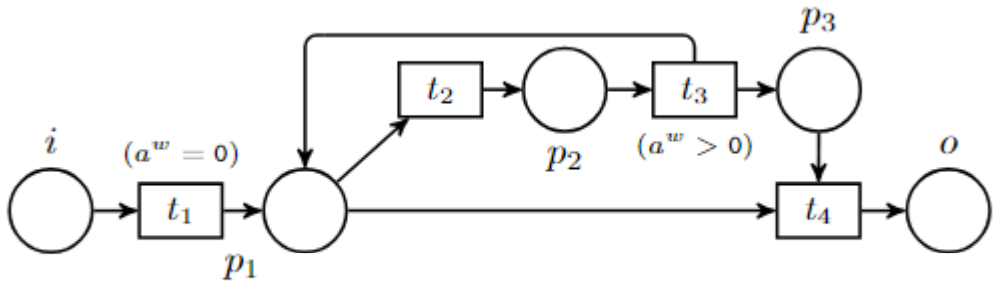
1. Make a model bounded.
2. Eliminate deadlocks and livelocks.
3. Eliminate dead transitions and isolated places.

Repair Approach

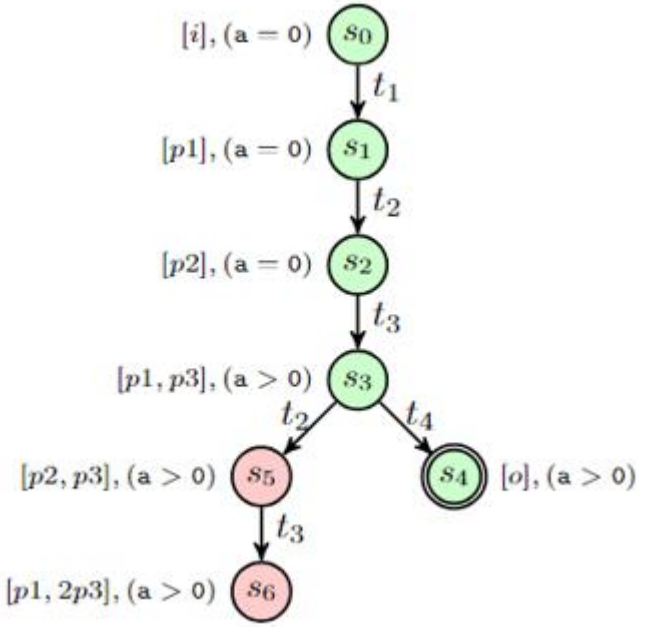


1. **Make a model bounded.**
2. Eliminate deadlocks and livelocks.
3. Eliminate dead transitions and isolated places.

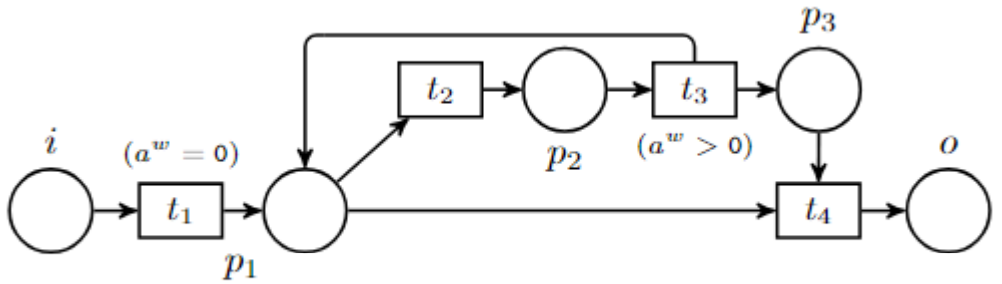
Repair Approach



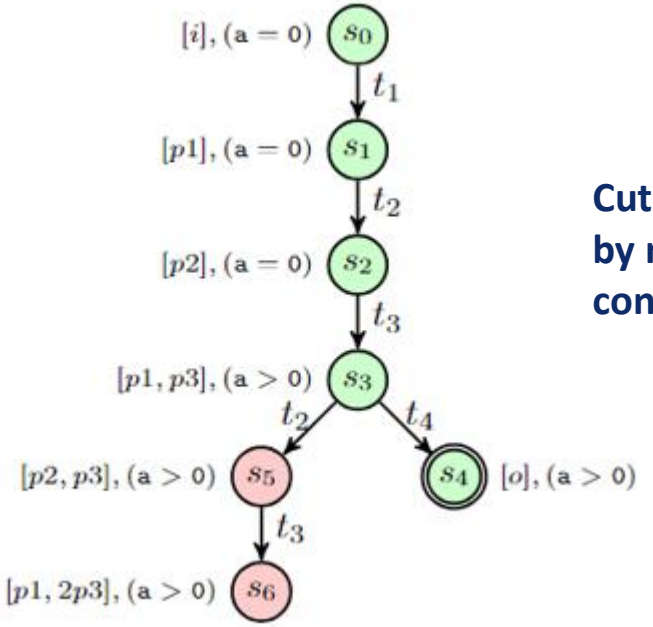
1. **Make a model bounded.**
2. Eliminate deadlocks and livelocks.
3. Eliminate dead transitions and isolated places.



Repair Approach

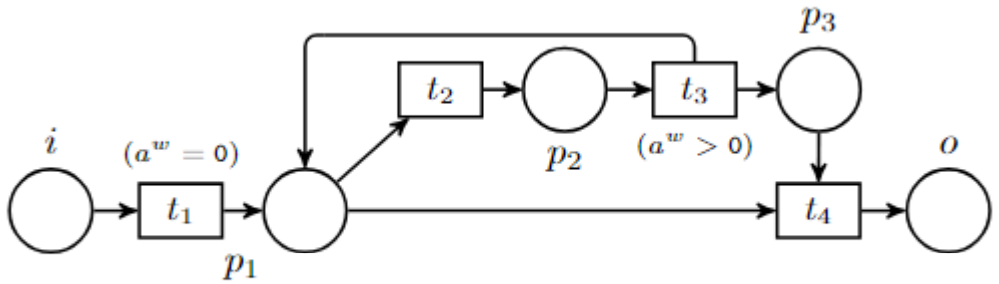


1. **Make a model bounded.**
2. Eliminate deadlocks and livelocks.
3. Eliminate dead transitions and isolated places.

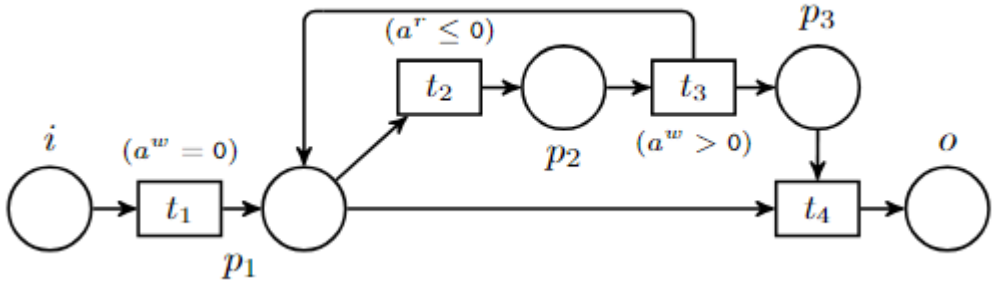


Cut off branches with red nodes by restricting the transition constraints

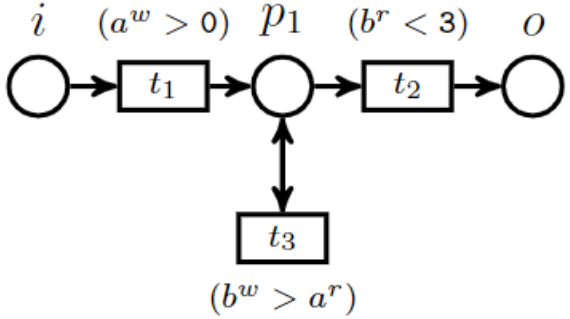
Repair Approach



1. **Make a model bounded.**
2. Eliminate deadlocks and livelocks.
3. Eliminate dead transitions and isolated places.



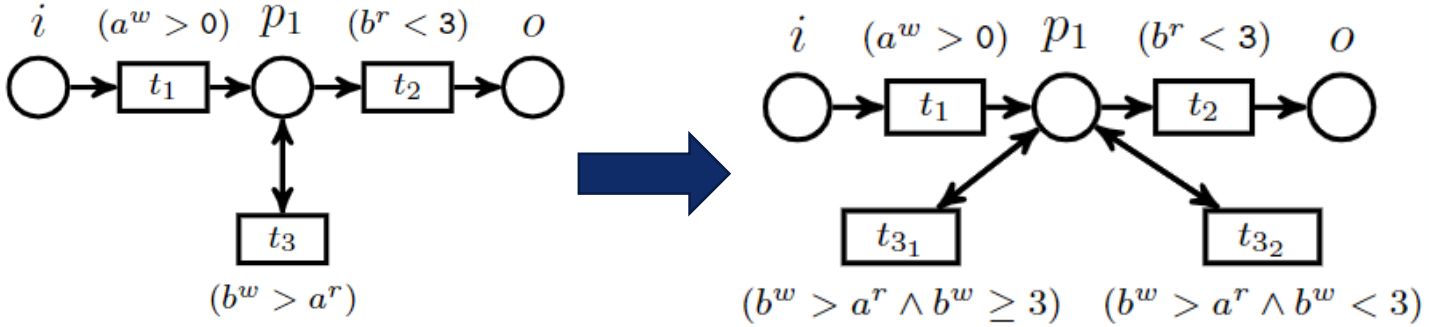
Repair Approach



1. Make a model bounded.
2. **Eliminate deadlocks and livelocks.**
3. Eliminate dead transitions and isolated places.

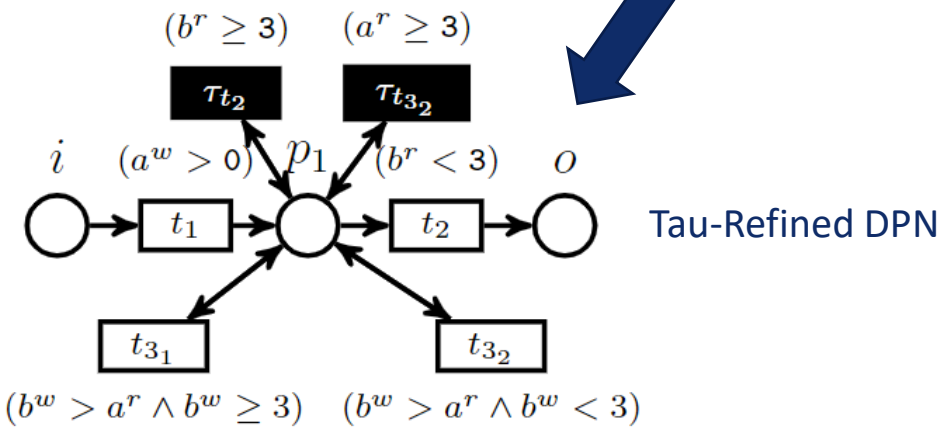
Repair Approach

1. Make a model bounded.
2. **Eliminate deadlocks and livelocks.**
3. Eliminate dead transitions and isolated places.



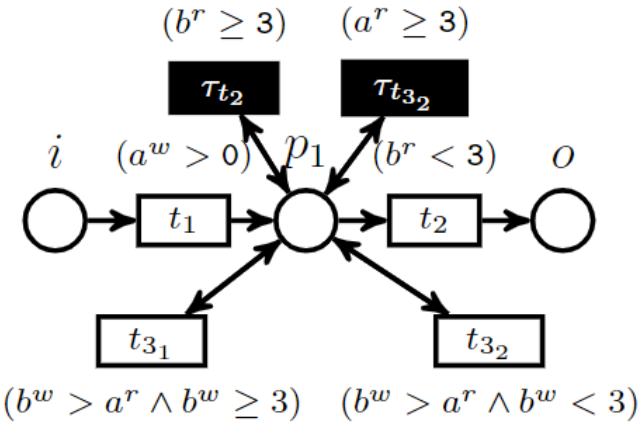
Source DPN

Refined DPN

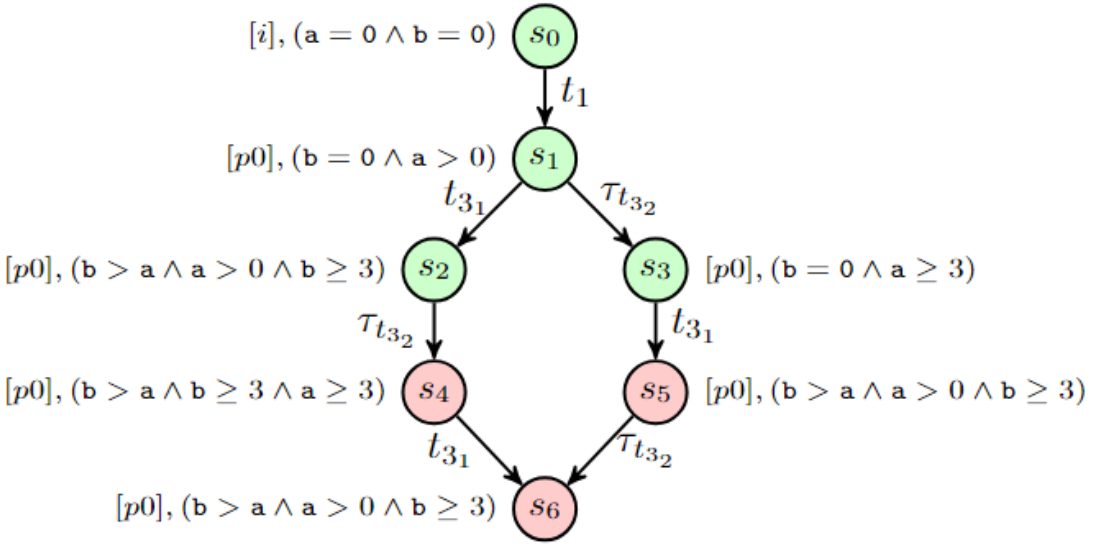


Tau-Refined DPN

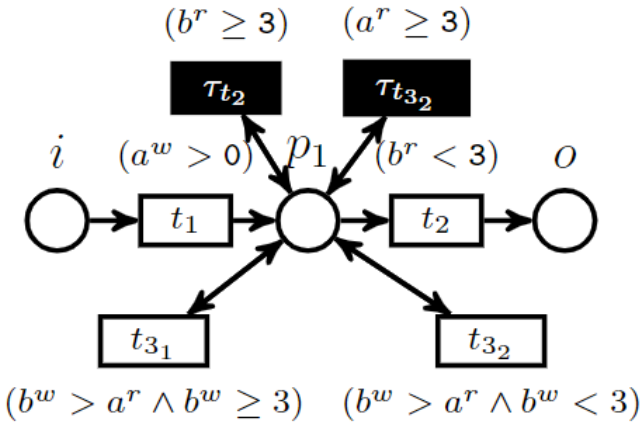
Repair Approach



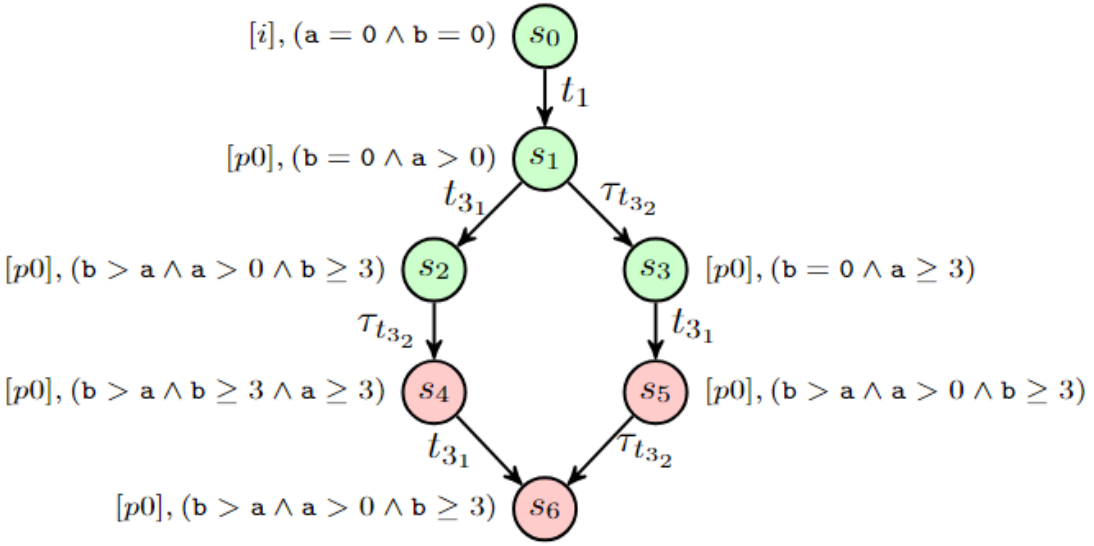
1. Make a model bounded.
2. **Eliminate deadlocks and livelocks.**
3. Eliminate dead transitions and isolated places.



Repair Approach

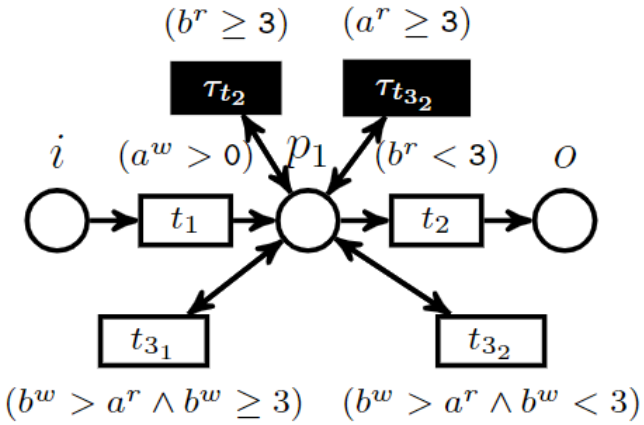


1. Make a model bounded.
2. **Eliminate deadlocks and livelocks.**
3. Eliminate dead transitions and isolated places.

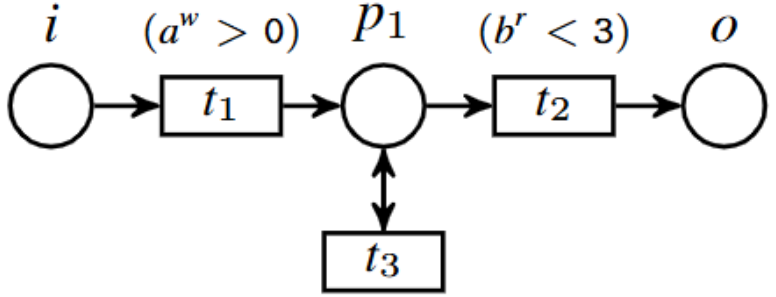


Cut off branches with red nodes by restricting the transition constraints

Repair Approach



1. Make a model bounded.
2. **Eliminate deadlocks and livelocks.**
3. Eliminate dead transitions and isolated places.



Lastly merge all the refined transitions back

Repair Approach

1. Make a model bounded.
2. Eliminate deadlocks and livelocks.
3. **Eliminate dead transitions and isolated places.**

Remove all the transitions that are not present in the final ARG.

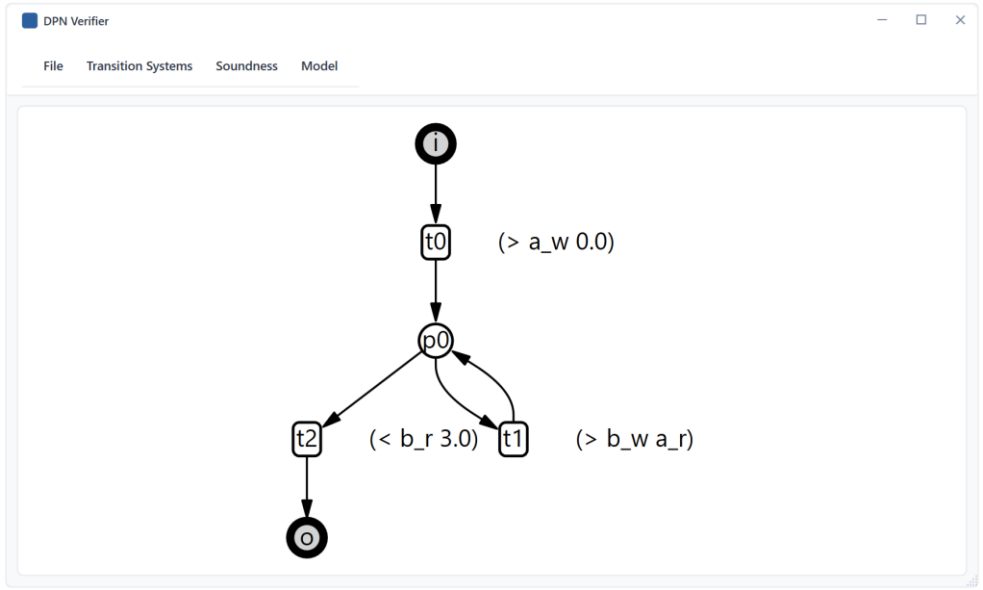
Remove isolated places & all the places that do not have tokens in any markings.

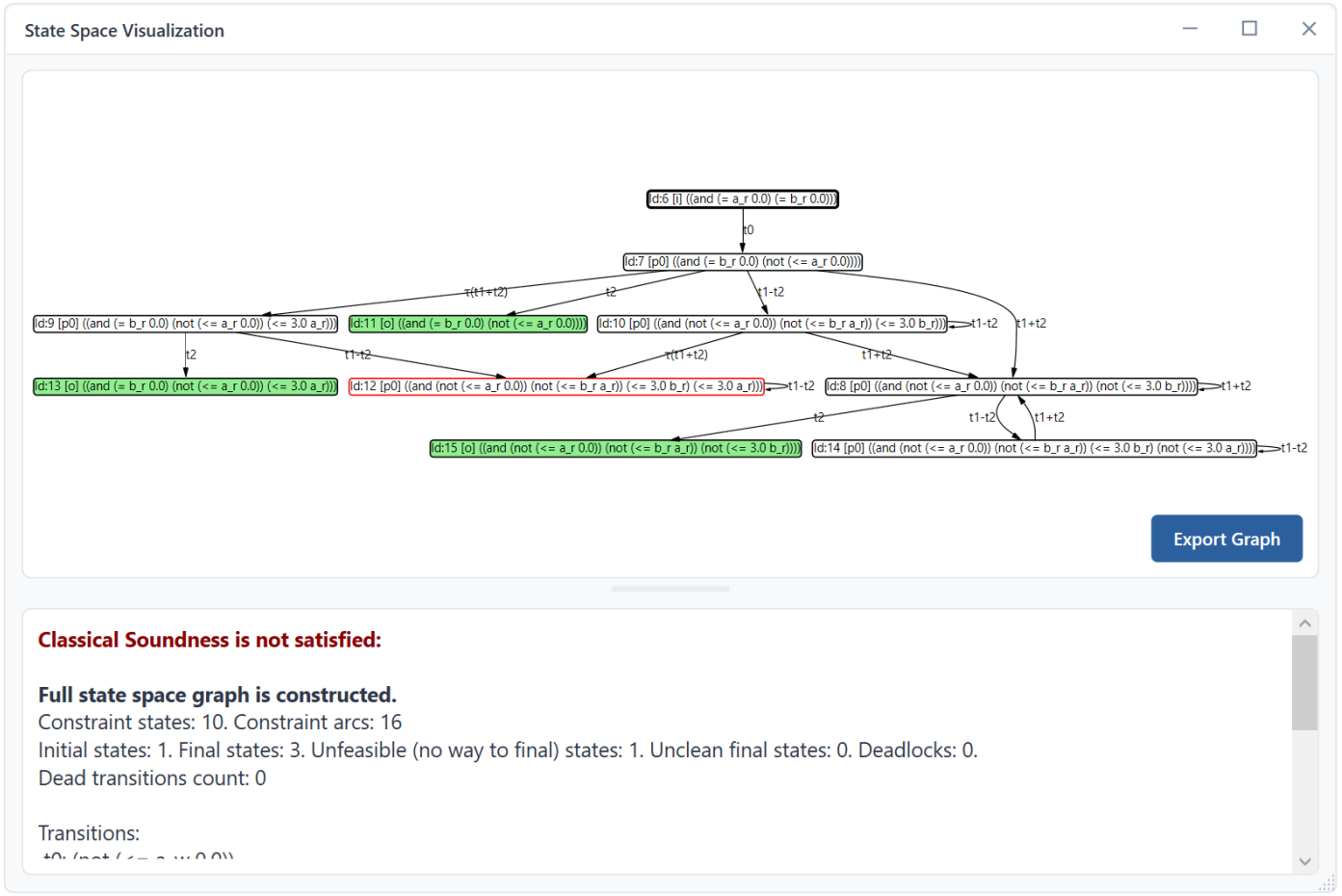
Repair Time

Model	*RepairDPN* Repair Time	*RepairDPN* Repair Steps	Algorithm [15] Repair Time	Algorithm [15] Repair Steps
Casino Example (Fig. 1)	169 ms	2	2.7 s	2
Livelock Example (Fig. 7)	203 ms	1	2.1 s	1
Unbounded Example (Fig. 10)	40 ms	1	-	-
Digital Whiteboard: Transfer [38]	143 ms	4	2.1 s	1
Package Handling [11]	4.8 s	0	6 s	0
Road Fines Mined [38]	1.9 s	1	24 s	1
Simple Auction [15]	318 ms	1	2.5 s	1

Intel Core I7-12700H

Prototype


DPN Verifier
File
Transition Systems
Soundness
Model
i
t0
(> a_w 0.0)
p0
t2
(< b_r 3.0)
t1
(> b_w a_r)
o


State Space Visualization
Id:6 [i] ((and (= a_r 0.0) (= b_r 0.0)))
t0
Id:7 [p0] ((and (= b_r 0.0) (not (<= a_r 0.0))))
Id:9 [p0] ((and (= b_r 0.0) (not (<= a_r 0.0)) (<= 3.0 a_r)))
Id:11 [o] ((and (= b_r 0.0) (not (<= a_r 0.0))))
Id:10 [p0] ((and (not (<= a_r 0.0)) (not (<= b_r a_r)) (<= 3.0 b_r)))
Id:13 [o] ((and (= b_r 0.0) (not (<= a_r 0.0)) (<= 3.0 a_r)))
Id:12 [p0] ((and (not (<= a_r 0.0)) (not (<= b_r a_r)) (<= 3.0 b_r) (<= 3.0 a_r)))
Id:8 [p0] ((and (not (<= a_r 0.0)) (not (<= b_r a_r)) (not (<= 3.0 b_r))))
Id:15 [o] ((and (not (<= a_r 0.0)) (not (<= b_r a_r)) (not (<= 3.0 b_r))))
Id:14 [p0] ((and (not (<= a_r 0.0)) (not (<= b_r a_r)) (<= 3.0 b_r) (not (<= 3.0 a_r))))
Export Graph
Classical Soundness is not satisfied:
Full state space graph is constructed.
Constraint states: 10. Constraint arcs: 16
Initial states: 1. Final states: 3. Unfeasible (no way to final) states: 1. Unclean final states: 0. Deadlocks: 0.
Dead transitions count: 0
Transitions:

Conclusion

Verifying soundness on DPN allows for a subtle investigation of each execution scenario helping to detect deadlocks and livelocks that could be hidden in other formalisms.

Our soundness verification technique allows to decrease the number of needed state space constructions, which helps to decrease the overall execution time.

The proposed repair algorithm incorporates the verification techniques and, by that, could repair a model rather fast. By exploiting coverability graph abstractions, the algorithm allows to repair some unbounded models.

Both the algorithms could be used to detect and eliminate errors both in manually constructed models and in the models automatically constructed from event logs.

Papers

Nikolai M. Suvorov, Irina A. Lomazova Verification of data-aware process models: Checking soundness of data Petri nets *// Journal of Logical and Algebraic Methods in Programming. 2024.* Vol. 138. Article 100953.

Suvorov N. M., Lomazova I. A. Soundness Correction of Data Petri Nets // IEEE Access. 2025. Vol. 13. P. 149142–149157.

Thanks for your attention!

Presenter: Nikolai M. Suvorov
nmsuvorov@edu.hse.ru
nmsuvorov@hse.ru