Strategy Synthesis for decision-aware process models

Paolo Felli
unibz

paolo.felli@unibz.it

August 28, 2020

Motivations

Many complex processes are described in terms of their constituent tasks, and how the resulting execution is affected by the **data** that these tasks manipulate.

Control-flow dimension + data dimension

These tasks operate over a finite set of variables with infinite domain, and their executability is guarded by conditions on the values of these variables.

Motivations

Many complex processes are described in terms of their constituent tasks, and how the resulting execution is affected by the **data** that these tasks manipulate.

Control-flow dimension + data dimension

These tasks operate over a finite set of variables with infinite domain, and their executability is guarded by conditions on the values of these variables.

These processes are nondeterministic

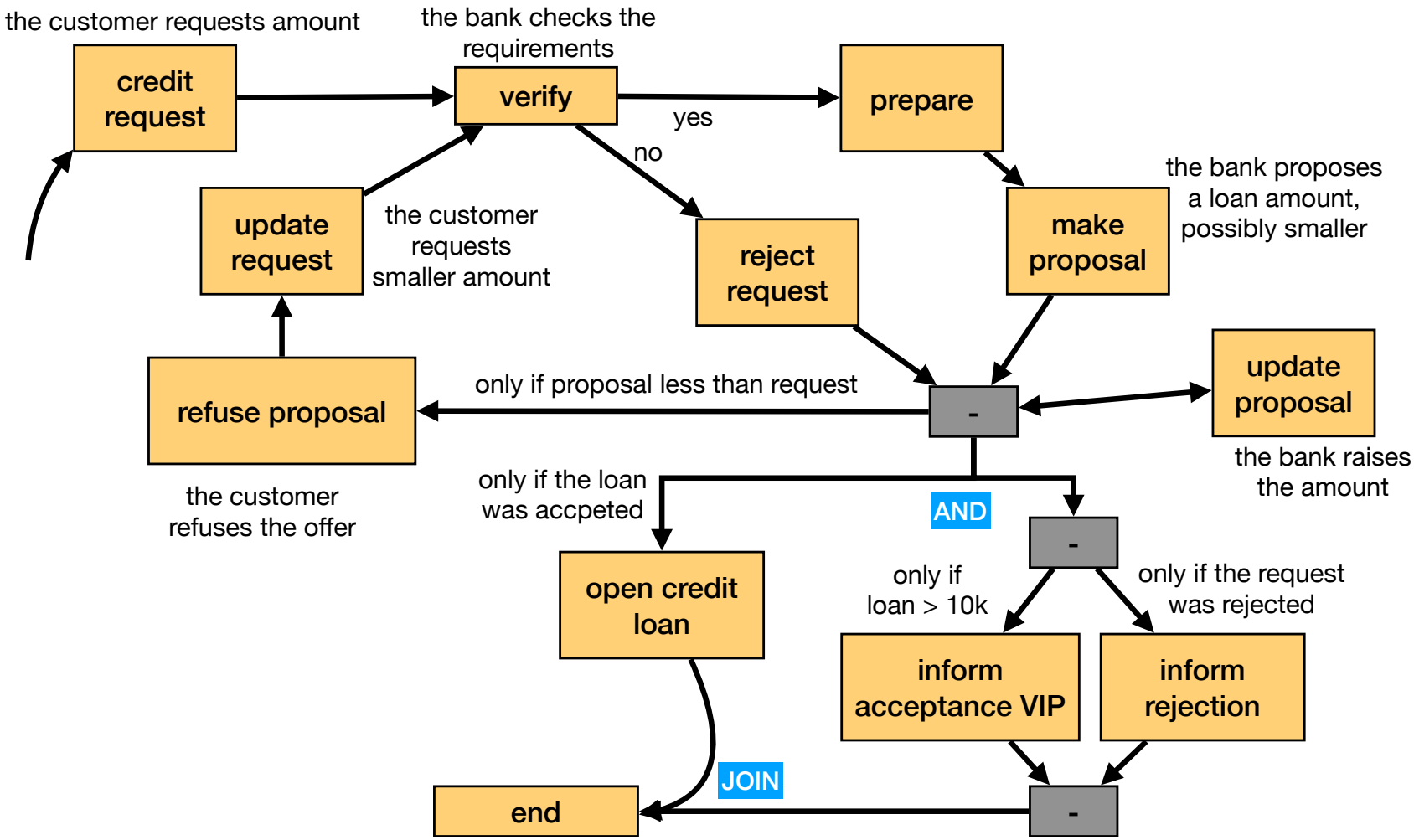
... when it comes to resolve decision points with multiple enabled conditions, or to choose which value to pick when updating the content of a variable.

Motivations

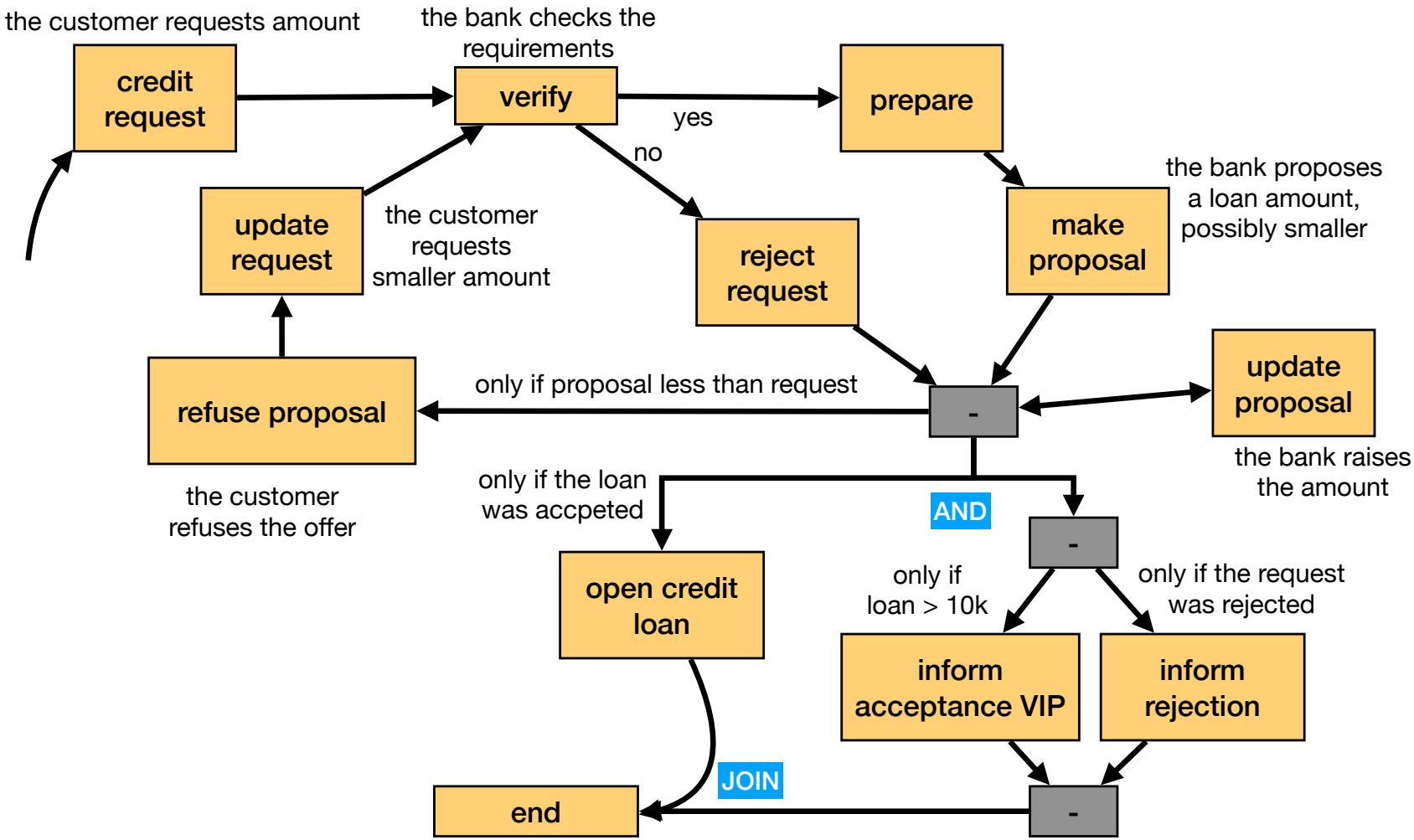
We need models of data-aware dynamic systems that are expressive enough to capture **decision-driven** processes. They should also be simple enough to be learned from event data by combining control-flow discovery and decision mining techniques.

And we need **computational** approaches for the analysis of both dimensions, to be able to assess the correctness of the process w.r.t. given specifications.

Processes with data and conditions (/decisions)



Processes with data and conditions (/decisions)



- Is the process correct?
- Which properties does it satisfy?

Correctness of processes with data and conditions

A naive approach might be to disregard data.

1. Remove data and conditions
2. Check the desired specifications, for instance:

Correctness of processes with data and conditions

A naive approach might be to disregard data.

1. Remove data and conditions
2. Check the desired specifications, for instance:

Classical notions of soundness:

- the end of the process is always reachable;
- no thread/branch is left active;
- each task can be executed in some execution.

Temporal constraints

E.g., expressed in Linear Temporal Logic on finite strings (LTLf), where traces are only constituted by sequences of task symbols.

LTL(f) allows to express linear properties of traces such as:

- eventually in the trace an task is executed;
- a deadlock is never reached;
- arbitrary patterns...

Issues

This would not be enough.

1. The analysis must be data/decision-aware

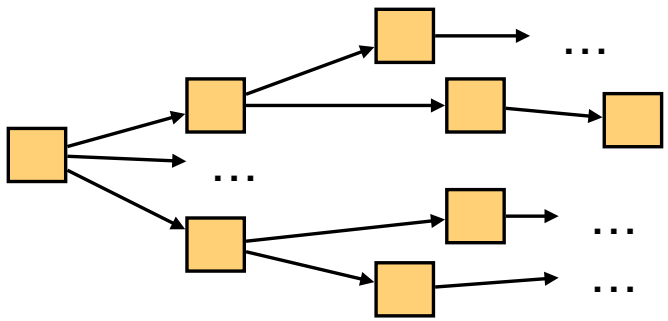
Since the execution of the process depends on the data that is written and tested, this sort of analysis is not sufficient!

Issues

And if we do consider the data dimension:

2. These processes are intrinsically infinite-state

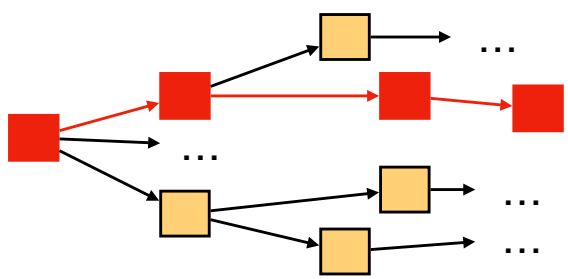
Tasks operate over a finite set of variables with infinite domain, and their executability is guarded by conditions on the values of these variables.



In general, we can always write fresh values, taken from an infinite domain.

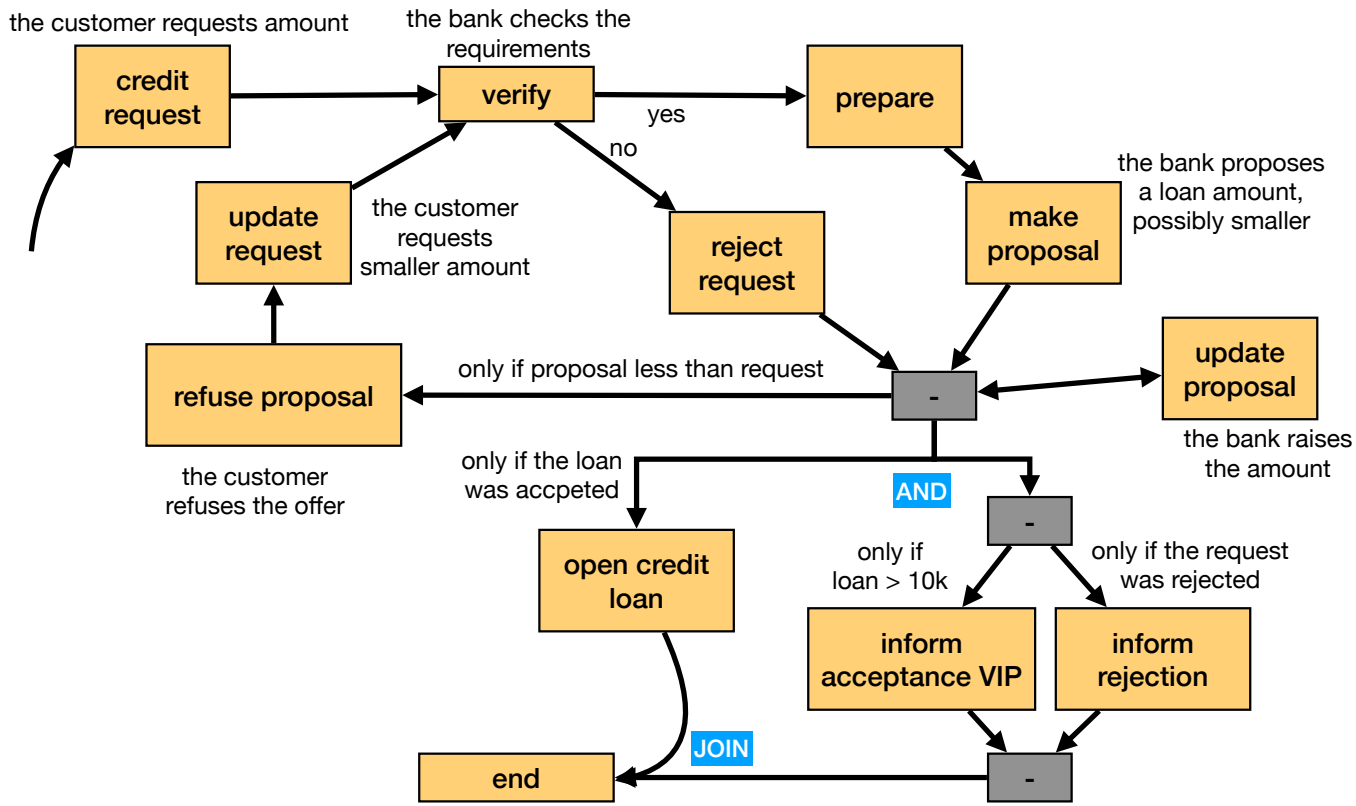
3. How can we adapt known verification/synthesis techniques for LTLf

We need to compute execution strategies that guarantee given properties.



Verification of data-aware properties

For instance, we may want to check whether **there exists a way** of receiving a VIP acceptance for a loan larger than the requested amount, after the request is initially denied, and executing "update request" at least once.



Data-aware processes with multiple actors

Moreover, processes typically involve **multiple actors** that control different sources of nondeterminism.

Verification approaches of these models often adopt the simplifying assumption that these actors cooperate.

Data-aware processes with multiple actors

Moreover, processes typically involve **multiple actors** that control different sources of nondeterminism.

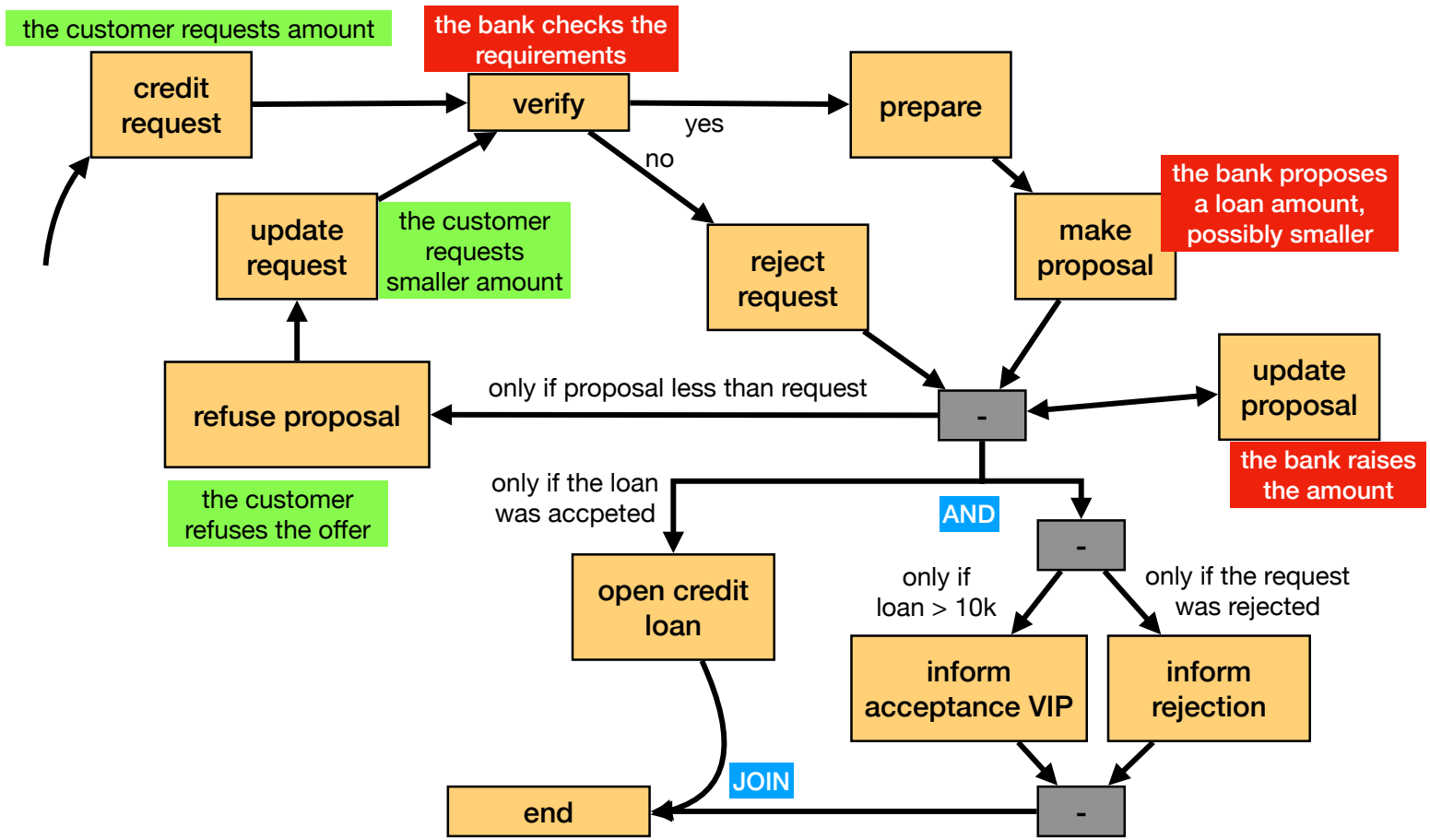
Verification approaches of these models often adopt the simplifying assumption that these actors cooperate.

Instead, we assume that each task and each data variable is controlled *by one actor*.

If then we adopt the perspective of **one** such actor, so that all the tasks that are not controlled by the actor are assumed to be controlled by an abstract antagonist controlling everything else, we might be interested in checking whether the actor can make their choices so that the induced executions of the process satisfy a given specification.

Data-aware processes with multiple actors

In the example, we can easily identify two actors:



If the actor we choose is the customer, can the customer select what to execute (and how write their variables) so that a property is true in the resulting executions?

The approach

1. We propose and adopt a simple model of data-aware processes, based on Petri nets
2. We define a specification language for data-aware processes, that allow to express requirements on the control-flow as well as simple data conditions
3. We define the notion of existence of strategies that satisfy a given formula
4. We adapt the usual reactive synthesis approach and captured the problem as a two-player adversarial DFA game, to automatically compute such strategies.

Step 1: data-aware process models: DPNs

We consider a set of data variables V (either booleans or in $\mathbb{R}$) and define:

$V^r = \{v^r \mid v \in V\}$ – denoting the current value of v
$V^w = \{v^w \mid v \in V\}$ – denoting the next value of v after a task is executed

A **constraint** is a simple comparison of:

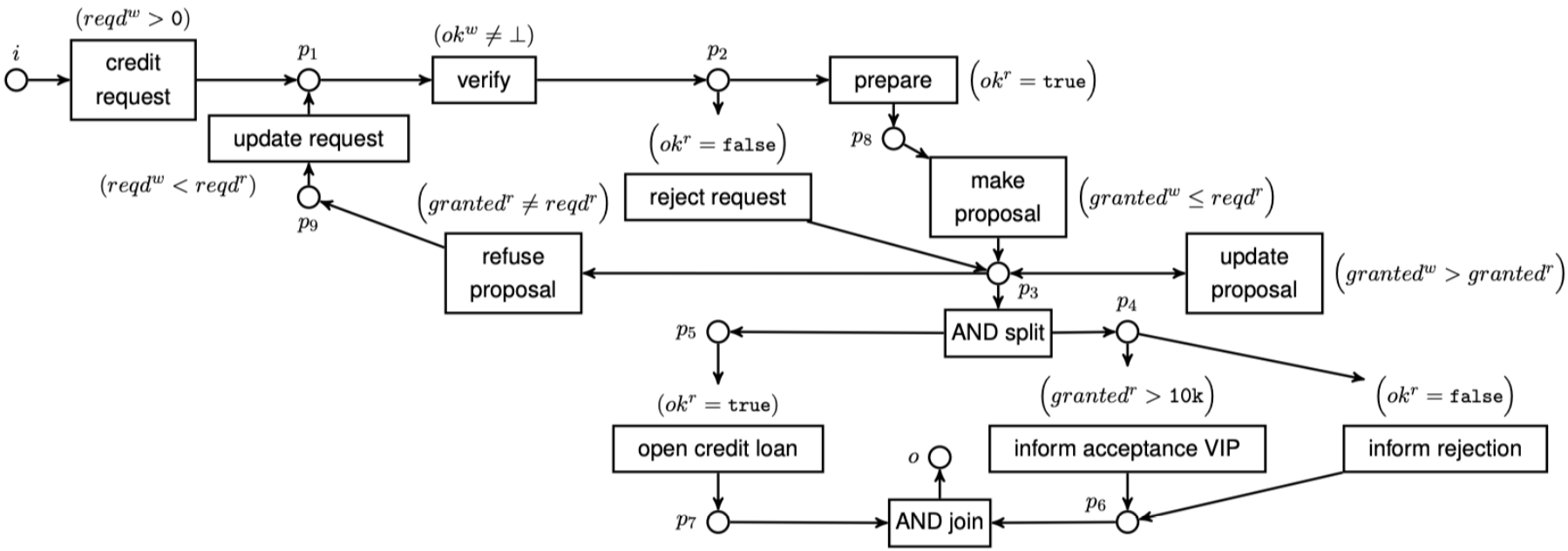
- A variable $v_1^r \in V^r$ with a constant or another variable $v_2^r \in V^r$;
  examples: $(x^r > 5)$, $(x^r <= y^r)$, $(z = \texttt{true})$ ...

- A variable $v_1^w \in V^w$ with a constant or a variable $v_2^r \in V^r$;
  examples: $(x^w > 5)$ or $(x^w <= y^r)$ ...

For simplicity, in these slides we assume only the domain $\mathbb{R}$ and the operators $\{<, >, =, \neq, \leq, \geq\}$, or the boolean domain (with $\{=, \neq\}$), but the approach works for finite domains and for domains that are dense (wrt the operators).

Step 1 (cont'd): DPNs

Then we represent a data-aware process as a **DPN** N: essentially, a **bounded** Petri net (for the control-flow) with all transitions associated to one constraints as before (for the data dimension).



We assume an initial and a final marking to be fixed.

Step 1 (cont'd): runs of DPNs

On DPNs, we define the notions of **runs**, i.e., runs of the DPN in which, at each step, we have the marking and a set of constraints that are true at that step.

$$\rho = (M_0, C_0) \xrightarrow{t_1,\beta_1} (M_1, C_1) \xrightarrow{t_2,\beta_2} \cdots \xrightarrow{t_n,\beta_n} (M_n, C_n)$$

A couple t, β is a *transition firing*: a transition in the net associated to a function that, informally, specifies how the variables are updated as the transition is fired.

example (the condition associated to `credit request` is $reqd^w > 0$)

$$([i], \{(reqd = 0), (ok = false), (granted = 0)\})$$

$$\xrightarrow{\texttt{credit request},\{\beta(reqd^w)=3999\}}$$

$$([p1], \{(reqd = 3999), (ok = false), (granted = 0)\})$$

In general, a DPN has infinite runs (in length and number).

Step 2: specification language

The runs of a DPN can be infinite, but in line with the intuition that "each process execution must eventually terminate", we define a specification language with a semantics defined on finite runs (analogous to LTLf).

Given a DPN N, consider the language with grammar:

$$\psi = \texttt{true} \mid C \mid M \mid \neg\psi \mid \psi_1 \wedge \psi_2 \mid \psi_1 \vee \psi_2 \mid \langle t \rangle \psi \mid \Diamond \psi \mid \Box \psi$$

where C is a set of constraints and M is a marking of N.

Step 2: semantics

$$\rho = (M_0, C_0) \xrightarrow{t_1,\beta_1} (M_1, C_1) \xrightarrow{t_2,\beta_2} \cdots \xrightarrow{t_n,\beta_n} (M_n, C_n)$$

Given a finite run ρ of a DPN N and a formula ψ, we say that ρ satisfies ψ, written $\rho \vDash \psi$, iff $\rho, 0 \vDash \psi$ according to the following semantics, where $i \in [0, \mathit{lenght}(\rho){-}1]$:

$\rho, i \vDash \texttt{true}$

$\rho, i \vDash C$ *iff* $C \cup C_i$ is satisfiable, with $\rho[i] = (b, C_i)$

$\rho, i \vDash M$ *iff* $\rho[i] = (M, C)$ for some C

$\rho, i \vDash \neg\psi$ *iff* $\rho, i \nvDash \psi$

$\rho, i \vDash \psi_1 \wedge \psi_2$ *iff* $\rho, i \vDash \psi_1$ and $\rho, i \vDash \psi_2$

$\rho, i \vDash \langle t \rangle \psi$ *iff* $\rho[i] \xrightarrow{t,\beta} \rho[i+1]$ and $\rho, i+1 \vDash \psi$

$\rho, i \vDash \Diamond\psi$ *iff* $\exists j$ s.t. $i \leq j \leq \mathit{last}(\rho)$ and $\rho, j \vDash \psi$

$\rho, i \vDash \Box\psi$ *iff* $\forall j$ s.t. $i \leq j \leq \mathit{last}(\rho)$ we have $\rho, j \vDash \psi$

Where the positions of a run ρ, is denoted by $\rho[i]$, for $i \in [0, \mathit{length}(\rho) - 1]$.

Step 3: strategies: satisfaction of formulae in a DPN

We call *terminal* those runs that end with a marking of the DPN that is final.

informally:

Given a DPN N and a formula ψ as before, ψ is true in N, iff every run is a prefix of a terminal run, and $\rho \vDash \psi$ for every terminal (thus finite) run ρ.

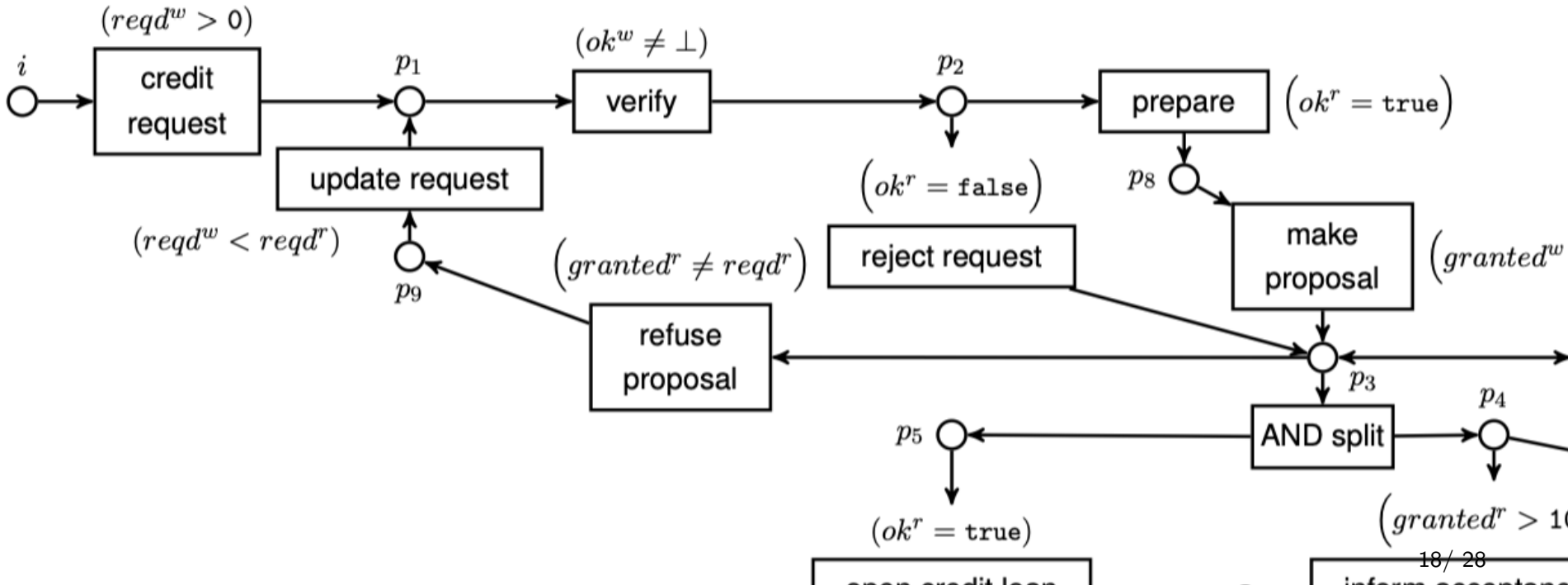
Note that we are requesting 2 things:

- runs can always potentially lead to the end of the process, and
- for terminal runs the formula is true.

Step 3: strategies

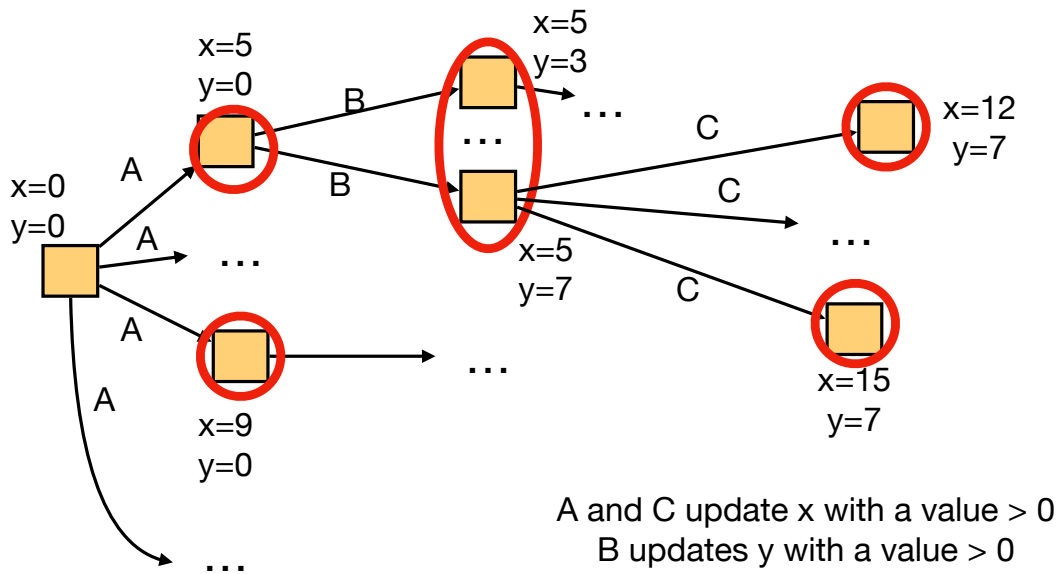
As already discussed, one actor typically does not control everything (they do not select at each step which task is executed and do not choose the values of all variables when these are updated/written).

Hence, when executing a strategy that specifies what an actor does at each step, we still need to face nondeterministic executions.



Step 3: strategies: controllability

Consider a simple example with two variables x, y, where x is controlled by the actor. Also, the actor controls the task A. The rest is controlled by other actors (the antagonist). We can compute the transition firings (thus the set of sets of successor states) that the actor can enforce.



Step 3: strategies that make a formula true

Based on this, we define strategies:

informally:

A strategy for an actor is a partial function ς which, given a finite run prefix of a DPN N, either returns a set of legal transition firings that can be **enforced** by the actor or it is undefined (if the run prefix is terminal).

Step 3: strategies that make a formula true

Based on this, we define strategies:

informally:

A strategy for an actor is a partial function ς which, given a finite run prefix of a DPN N, either returns a set of legal transition firings that can be **enforced** by the actor or it is undefined (if the run prefix is terminal).

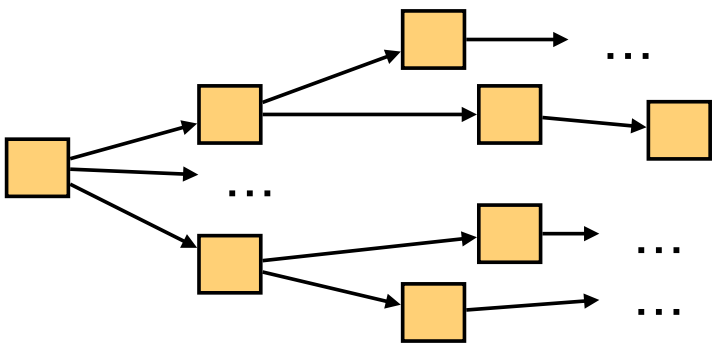
informally:

We say that a strategy ς for an actor satisfies a given formula iff the formula is true in N, but **without** considering all runs of N that are not enforced by ς.

Step 3: strategies that make a formula true

However:

Recall issue 2: These processes are intrinsically infinite-state

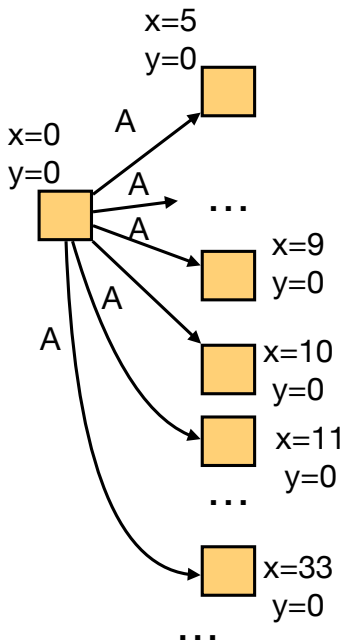


In general, we can always write fresh values, taken from an infinite domain.

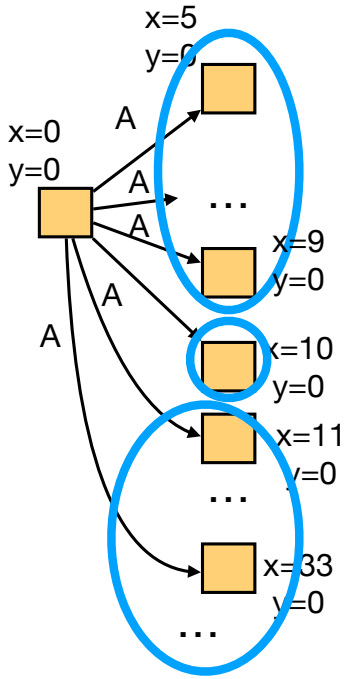
Hence in step 4 (the final synthesis step), we adopt an abstraction technique.

Step 4: preparation: Interval abstraction

We abstract all possible runs of the net as a finite-state transition system, adopting a simple interval abstraction on the possible values of all variables.



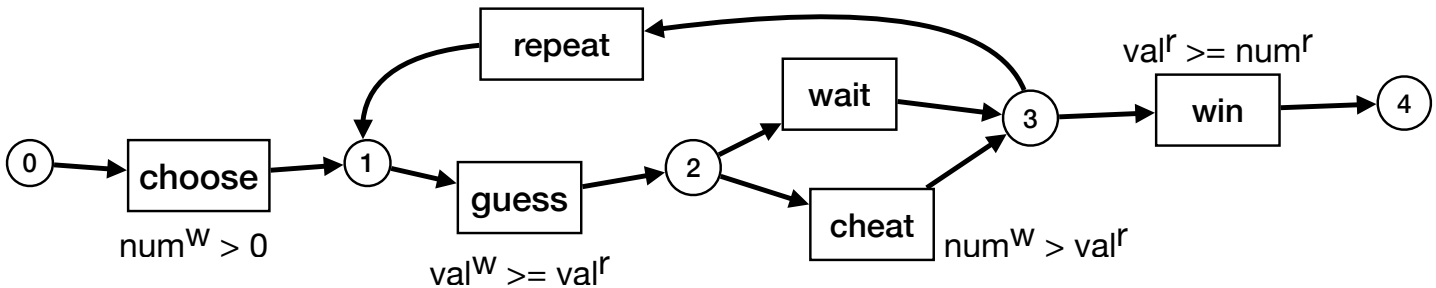
Assume that the only value of x that is ever tested, in any decision point in the process, is 10



These abstractions are faithful w.r.t. our specification language: a formula is true in a DPN iff it is true in its abstraction.

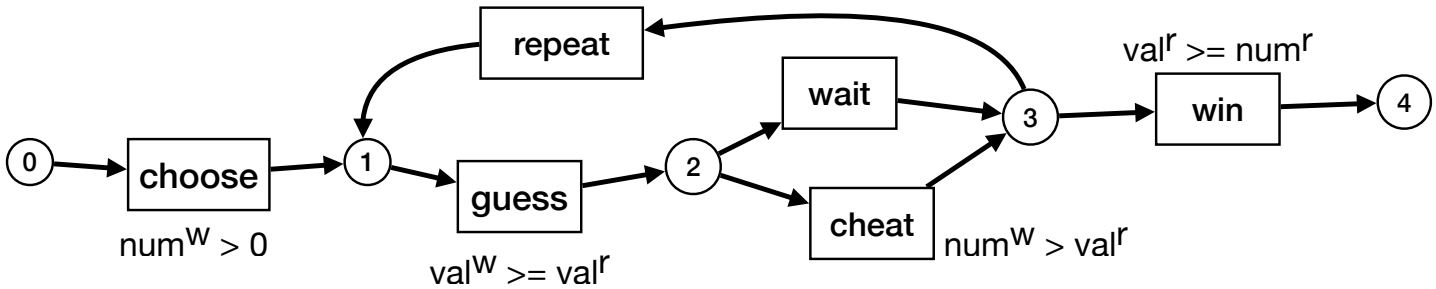
Step 4: preparation: Interval abstraction: example

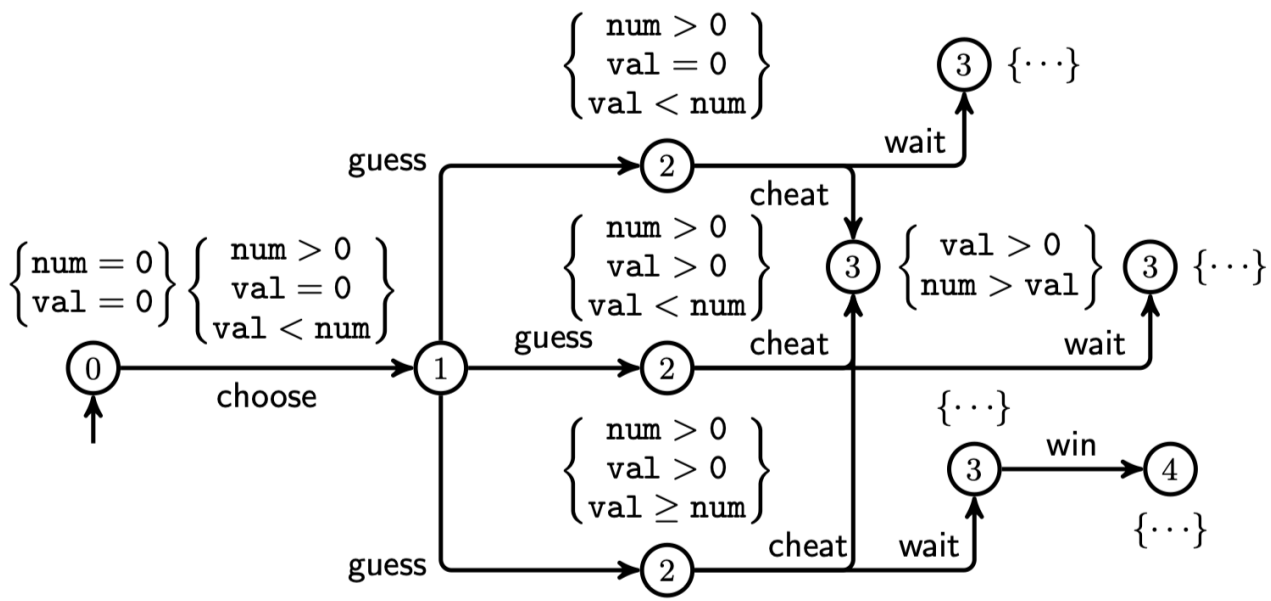
A number is chosen, then a guess is made. If the guess is equal or larger than the chosen number, the game is won. The transition cheat makes the current guess always wrong. All initial values are 0.



Step 4: preparation: Interval abstraction: example

A number is chosen, then a guess is made. If the guess is equal or larger than the chosen number, the game is won. The transition cheat makes the current guess always wrong. All initial values are 0.





Our abstraction guarantees that only finitely many states can be generated.

Step 4: synthesis

We follow a quite standard, automata-based approach for synthesis:

- We compute the deterministic finite-state automaton (DFA) for the formula;
- We abstract the DPN and trivially transform the abstraction into a DFA;
- We compute a DFA game arena by computing their (special) product, in which the satisfiability of combinations of constraint sets is checked.

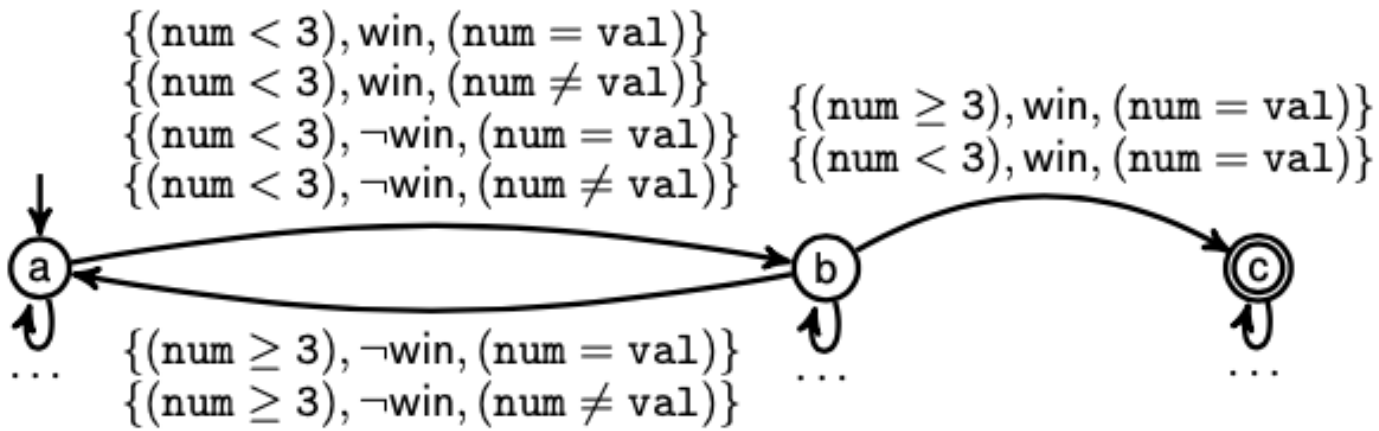
We then resolve the DFA game, which is **finite-state**: by applying a backwards fixpoint computation from terminal states, we compute the portion of the game that can be forced by the actor. If the initial state is in such set, then a strategy exists and we can compute it.

Step 4: synthesis: example

The specification formula, requiring the chosen real to be smaller than 3 and next (by executing the task `win`) the guess to be exact.

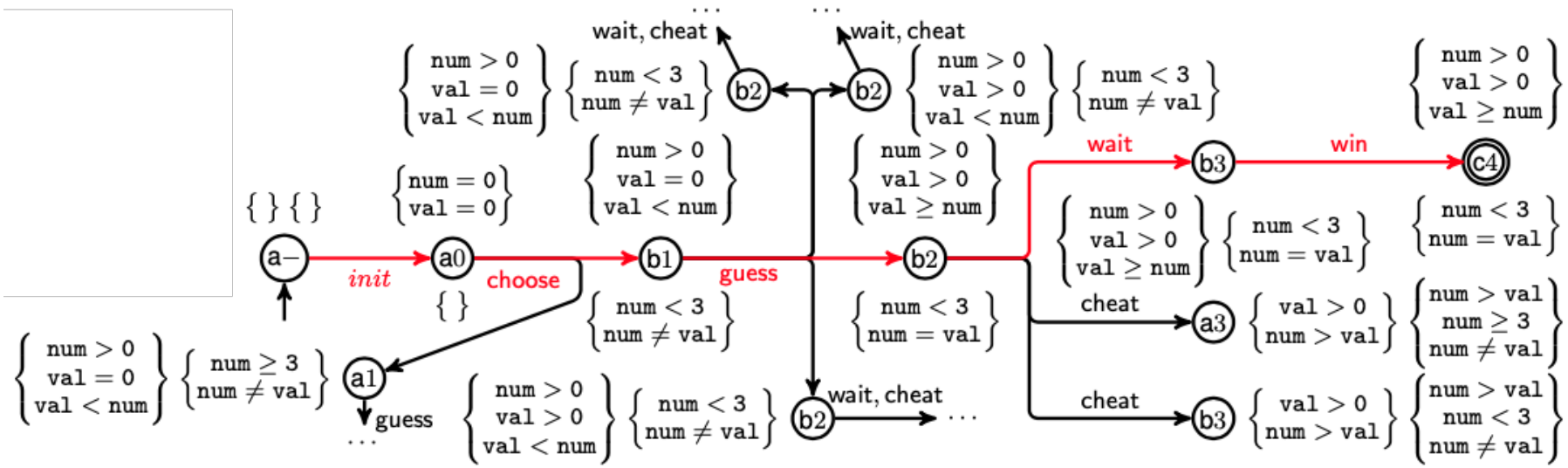
$$\psi = \Diamond((\mathtt{num} < 3) \wedge \langle \mathsf{win} \rangle (\mathtt{val} = \mathtt{num}))$$

The corresponding DFA:



Step 4: synthesis: example

Below, in red, it is shown a possible winning strategy for the DFA game, from which we can show how to extract a strategy for the initial DFA.



This shows that the game can be won by an actor if the variables `num` and `val`, and the tasks `wait` and `cheat`, are controlled by the actor.

Complexity

Given a DPN N and a formula ψ, synthesizing a strategy that guarantees ψ on N is exponential in N (due to the size of its abstraction) and doubly exponential in ψ (first, compute a NFA, then transform it into a DFA).

The hardness is given by strategy synthesis for LTLf, which corresponds to a special case of this problem.

Conclusions

We have developed an automata-based technique for computing winning strategies for data-aware processes for temporal specifications that also capture constraints on the data that these systems manipulate.

We achieved this by combining interval-based data abstraction techniques with standard automata-based constructions for verification.

The novelty is not related to the use of automata-based techniques for two-player adversarial games, but in the enrichment of these techniques with a data-aware feature.

The construction used in this paper can be directly implemented.

References.
[1] Strategy Synthesis for Data-Aware Dynamic Systems with Multiple Actors (to appear). M. de Leoni, P. Felli and Marco Montali. 17th International Conference on Principles of Knowledge Representation and Reasoning (KR 2020).
[2] Soundness Verification of Decision-Aware Process Models with Variable-to-Variable Conditions. P. Felli, M. de Leoni, M. Montali. 19th International Conference on Application of Concurrency to System Design (ACSD 2019).
[3] A Holistic Approach for Soundness Verification of Decision-Aware Process Models. M. de Leoni, P. Felli, M. Montali. 37th International Conference on Conceptual Modeling (ER 2018).