Verification of Coverability in Positive Interactive Datalog Programs

Francesco Di Cosmo[*]

Free University of Bozen-Bolzano, Italy

Abstract

We study the verification of coverability for Positive Interactive Datalog Programs (posIDP), a fragment of Communicating Datalog Programs (CDPs) where the programs are positive, i.e., make no use of negation. CDPs are a message passing system model grounded in logic programming, where nodes update their configuration by running a Datalog$^{\neq}$ program while sharing database tuples and receiving information from external services and/or users. While CDP verification has been extensively studied in the last few years, studies on CDP formal verification disregarded the role of negation in the node programs. In this paper, we study the impact on verification of positive programs when no meaningful use of negation is made. We show that the coverability problem remains undecidable despite the monotonicity of Datalog. We discuss the results in the framework of the Well Structured Transition System theory, motivating the introduction of novel semantics for data-aware processes that arguably enjoy better verification properties.

Keywords

Distributed Systems, Formal methods, Graph Minor, Parthood, WQO,

1. Introduction

Communicating Datalog Programs (CDPs) are a model of distributed computation where computational units run a data-aware process while sharing messages and receiving inputs from external users/services [1]. They are declarative data-centric, interactive, and message passing systems. Declarative and data-centric because CDP computational nodes repeatedly perform local computations by running a Datalog-like query (with stratified negation) over their information sources (primarily its local memory), which in turn are relational databases (DBs). Interactive because they model some degree of interaction with external users and/or external services. Message passing because the local computation steps of CDP nodes are triggered by the reception of a message and result in the sending of several messages on a network of imperfect channels among the nodes.

These features make CDPs interesting in several areas. First, they are tightly related with, and offer new perspectives on, well studied computational models of communication and concurrency, such as Communicating Finite State Machines [2], Lossy Channel Systems [3], Data Communicating Processes [4], and various forms of Petri Nets enriched with data [5, 6, 7]. Second, CDPs sit in the family of Declarative Networking models [8, 9, 10], where declarative query languages are used to concisely program and analyze networking protocols and services. Third, CDPs can be used for data-aware Business Process Management, where declarativity and data-centricity allow for a high level of expressiveness while avoiding the traditional abstraction of data [11]. Finally, the interactivity of CDPs makes them reminiscent of Answer Set Solvers that allow for multi-shot solving and interaction with scripts (e.g., Clingo [12] and Telingo [13]).

Starting from [14], formal verification of CDPs has been extensively studied. CDP verification is challenging, since interactivity and communication make CDPs infinite state systems. In general, even simple problems, e.g., control-state reachability, are undecidable [15]. Nevertheless, several CDP fragments enjoying decidability have been detected. For example, the constraint of data-boundedness,

CILC 2025: 40th Italian Conference on Computational Logic, June 25–27, 2025, Alghero, Italy
[*]Corresponding author.
frdicosmo@unibz.it (F. Di Cosmo)
0000-0002-5692-5681 (F. Di Cosmo)

where the capacity of the channels and the size of the active domains of the DBs involved in the computation are bounded (even if still from an infinite domain), yields PSPACE-completeness for CDP-CTL [15, 1], a rich specification language that mixes First Order (FO) logic, to query the node states, with Computation Tree Logic (CTL), to analyze its temporal evolution. The main drawback of state-boundedness is that it is a semantic, undecidable property, so that the undecidability of verification is discharged to the undecidability of recognizing data-bounded CDPs [16]. This motivated further works to lift state-boundedness while retaining decidability. For example, [17, 18] show how undecidability (in particular of termination) stems not only from unboundedness but from the interplay of unboundedness and the mechanisms of CDP nodes to recall the previous configuration. More recently, [19] shows that decidability of model checking of coverability-based fragments of CDP-CTL can be gained when the channels are unbounded but the messages range over an at most unary signature.

In this paper, we further study the verification of CDPs, by considering the limited setting of single-node CDPs running positive Datalog-like programs and using communication only as a mechanism for triggering computation steps (via a self-loop channel). Since, in this setting, communication is negligible and only interactivity is relevant, we call the resulting model Positive Interactive Datalog Programs (posIDP). posIDP appear promising for verification, since it is known that positive Datalog queries are monotone and, as a rule of thumb, monotonicity is related to better computational properties. In the area of verification, this principle is formalized by the theory of Well Structured Transition Systems (WSTS) [20], which provides general algorithms for the verification of properties like coverability on models that enjoy, among some other property, the notion of compatibility, which recasts monotonicity over transition systems. Despite the good properties of positive Datalog, we show that even the basic problem of coverability of propositional target configurations remains undecidable on posIDP. This motivates a discussion on why WSTS theory is not applicable on posIDP and, more in general, on data-aware processes. This leads to the proposal of new CDP semantics, arguably related to better verification properties. Our main technical contributions are undecidability results for posIDP coverability, irrespective of boundedness conditions on the information provided by external services. Moreover, we motivate and discuss a new conjecture about the decidability of a variant of coverability for posIDP whose semantics is based on the notion of graph-minor.

In Sec. 2 we provide preliminaries on WSTSs and 2CM. In Sec. 3 we define posIDP and the technical problem studied in the paper. In Sec. 4 we prove the undecidability results. In Sec. 5 we discuss why WSTS theory cannot be applied on posIDP and conjecture decidability for posIDP based on graph-minors. Finally, in Sec. 6, we draw the conclusions.

2. Preliminaries

2.1. WSTS

We introduce WSTSs as in [20].

Definition 1. A *quasi-order* (QO) $\preceq$ on a set A is a reflexive and transitive binary relation on A. The QO $\preceq$ is a *WQO* if, for each infinite sequence $(a_i)_{i \in \mathbb{N}}$ over A, there are $i, j \in \mathbb{N}$ such that $i < j$ and $a_i \preceq a_j$.

Given a QO $\preceq$ on A, for each $a_1, a_2 \in A$, we write $a_1 \prec a_2$ if $a_1 \preceq a_2$ and $a_1 \neq a_2$, and write $a_2 \succeq a_1$ (respectively $a_2 \succ a_1$) if $a_1 \preceq a_2$ ($a_1 \prec a_2$).

Definition 2. Given a QO $\preceq$ on A, the *up-ward closure* $\uparrow\alpha$ of a subset $\alpha \subseteq A$ is the set $\{a \in A \mid \exists a' \in \alpha : \ a' \preceq a\}$. The set α is *upward-closed* if $\alpha = \uparrow\alpha$. A *base* for an up-ward closed set α is a set $\alpha' \subseteq \alpha$ such that, for each $a \in \alpha$, there is some $a' \in \alpha$ such that $a' \preceq a$.

It is well-known that, under any WQO, each upward-closed set has a finite base [21].

Definition 3. A *transition system* (TS) $\mathcal{T}$ is a pair $\mathcal{T} = \langle \text{Conf}, \to \rangle$ where Conf is a (possibly infinite) set of configurations and $\to$ is a binary relation on Conf. We denote by $\to^*$ the reflexive and transitive closure of $\rightarrow$. Given a QO $\preceq$ on Conf, the TS $\mathcal{T}$ is *compatible* (*strongly compatible*) with $\preceq$ if, for each $c_1, c_2, c_3 \in$ Conf such that $c_1 \succeq c_2 \rightarrow c_3$, there is some $c_4 \in$ Conf such that $c_1 \rightarrow^* c_4 \succeq c_3$ (respectively $c_1 \rightarrow c_4 \succeq c_3$). A *WSTS* is a tuple $\langle \text{Conf}, \rightarrow, \preceq \rangle$ where $\langle \text{Conf}, \rightarrow \rangle$ is a TS, $\preceq$ is a WQO over Conf, and $\mathcal{T}$ is compatible with $\preceq$.

Definition 4. Given a TS $\mathcal{T} = \langle \text{Conf}, \rightarrow \rangle$ and a QO $\preceq$ over Conf, the *$\preceq$-coverability* problem for $\mathcal{T}$ is the following decision problem:
Input Two configurations $q_{init}, q_{target} \in$ Conf.
Output Whether there is a configuration $q \in$ Conf such that $q_{init} \rightarrow^* q \succeq q_{target}$.

This problem is decidable if the WSTS enjoys a couple of effectiveness assumptions. Specifically, given a WSTS $\mathcal{T} = \langle \text{Conf}, \rightarrow, \preceq \rangle$ and a set $\alpha \subseteq$ Conf, we denote by $\text{Pred}(\alpha)$ the set of all *predecessors* of configurations in α, i.e., $\text{Pred}(\alpha) = \{q \in \text{Conf} \mid \exists q' \in \alpha : \ q \rightarrow q'\}$. We say that a WSTS has the *Effective Pred-Basis* (EPB) property if there is an algorithm such that, given a configuration q, it returns a finite basis of $\uparrow\text{Pred}(\uparrow\{q\})$. Moreover, we say that a QO $\preceq$ over A is decidable if there is an algorithm such that, given $a_1, a_2 \in A$, it decides whether $a_1 \preceq a_2$.

Theorem 1. *The $\preceq$-coverability problem is decidable for WSTSs $\langle$Conf, $\rightarrow, \preceq\rangle$ with EPB and decidable $\preceq$.*

2.2. 2-Counter Machines

A 2-Counter Machine (2CM), also called Minsky Machine, is a Finite State Automaton (FSA) whose transitions can handle two counters, by performing increments and conditional decrements.

Definition 5. A *2CM K* is a tuple $K = \langle Q, \Delta, q_{init}, q_{fin} \rangle$ where Q is a non-empty finite set of states, $q_{init}, q_{fin} \in Q$, and Δ is a finite set of instructions such that, for each $\delta \in \Delta$, either $\delta = (q, +, i, q')$, called *increment transition*, or $\delta = (q, -, i, q', q'')$, called *conditional decrement transition*, for some $q, q', q'' \in Q$ and $i \in \{1, 2\}$.[1] The 2CM K is *deterministic* if, for each two transitions $\delta_1, \delta_2 \in \Delta$, the first components of δ_1 and δ_2 are distinct.

2CM configurations track the FSA state and the numbers in the counters.

Definition 6. The set Conf of *configurations* of a 2CM $K = \langle Q, \Delta, q_{init}, q_{fin} \rangle$ is the set $Q \times \mathbb{N}^2$. The *transition system of K* is the directed graph (Conf, $\rightarrow$) such that $\rightarrow = \bigcup_{\delta \in \Delta} \rightarrow^{\delta}$ where, for each $q_1, q_2 \in Q$, $\delta \in \Delta$, and $i \in \{0, 1\}$, we have $\langle q, n_0, n_1 \rangle \rightarrow^{\delta} \langle q', n_0', n_2' \rangle$ if $n_{1-i}' = n_{1-i}$ and:

1. $\delta = \langle q, +, i, q' \rangle$, and $n_i' = n_i + 1$,
2. $\delta = \langle q, -, i, q', q'' \rangle$, $n_i > 0$, and $n_i' = n_i - 1$, or
3. $\delta = \langle q, -, i, q'', q' \rangle$, and $n_i = 0 = n_i'$.

The 2CM *termination problem* asks, given a 2CM $\langle Q, \Delta, q_{init}, q_{fin} \rangle$, whether $q_{init} \rightarrow^* q_{fin}$. It is well known that termination of deterministic 2CM is undecidable [22]. We assume, without loss of generality that, for each 2CM $K = \langle Q, \Delta, q_{init}, q_{fin} \rangle$, there is no instruction in Δ with q_{fin} in its first component, that is, once q_{fin} is reached, the computation terminates.

3. PosIDP

In this section we define posIDP. Because this CDP fragment features a single node, essentially no communication, and only positive programs, it enjoys the concise formalization below (cf. definition of general CDPs in [1]).

[1]The components of δ indicate, respectively: the state on which the transition fires; the type of the transition; the counter on which transition operates; the state reached after the increment or, if the counter is not zero, the decrement; the state reached after the decrement when the counter is zero.

3.1. Positive D2C

D2C is a Datalog-like query language specialized to the distributed and interactive setting of CDPs. Here we define its positive fragment. We assume that the reader is familiar with Datalog and relational databases (see [23] for an introduction). We work with DBs over a fixed infinite domain Δ of constants and denote the active domain of a DB D by $\Delta(D)$.

Given a relational signature $\mathcal{S} = \{S_1/n_1, \ldots, S_k/n_k\}$, we work with pair-wise disjoint relational signatures$\mathcal{S}^{pre} = \{S_1^{pre}/n_1, \ldots, S_k^{pre}/n_k\}$, $\mathcal{S}^{snd} = \{S_1^{snd}/n_1, \ldots, S_k^{snd}/n_k\}$, and $\mathcal{S}^{rcv} = \{S_1^{rcv}/n_1, \ldots, S_k^{rcv}/n_k\}$ of fresh symbols. These are used to refer to facts stored in the current state of the node ($\mathcal{S}$), the previous state ($\mathcal{S}^{pre}$), outgoing messages ($\mathcal{S}^{snd}$), and incoming message ($\mathcal{S}^{rcv}$). We lift this notation also to DBs, i.e., given a DB D over $\mathcal{S}$, $\mathcal{S}^{pre}$, $\mathcal{S}^{snd}$, or $\mathcal{S}^{rcv}$), we denote by D^{plain}, D^{pre}, D^{snd}, D^{rcv} the DB obtained by substituting each fact $S^{\alpha}(\mathbf{a})$ of D, where α is either the empty string or $\alpha \in \{pre, snd, rcv\}$, by the fact $S(\mathbf{a})$, $S^{pre}(\mathbf{a})$, $S^{snd}(\mathbf{a})$, or $S^{rcv}(\mathbf{a})$ respectively.

Definition 7. A D2C signature is a tuple $\Lambda = \langle \Delta_{spec}, \mathcal{S}, \mathcal{I}, \mathcal{C} \rangle$ where $\Delta_{spec} \subset \Delta$ is a finite set of constants, $\mathcal{S}$, $\mathcal{I}$, and $\mathcal{C}$ are pair-wise disjoint signatures, called, respectively *state*, *input*, and *channel* signatures, and $\texttt{start}/0 \in \mathcal{C}$.

In this section, we work with a fixed signature $\Lambda = \langle \Delta_{spec}, \mathcal{S}, \mathcal{I}, \{start/0\} \rangle$. A positive D2C (posD2C) rule is a Datalog (with inequalities) rule extended with dedicated symbols for sending messages (in the head), receiving messages and inputs (in the body), and querying the previous state (in the body). Let $\mathcal{V}$ be an infinite set of variables.

Definition 8. A posD2C rule over Λ is a formula '$\ H$ **if** $B_1, \ldots, B_n, c_1, \ldots, c_m$.' where:

1. H, called the *head*, is an FO atomic formula over $\mathcal{V}$, Δ_{spec}, and $\mathcal{S} \cup \mathcal{C}^{snd}$,
2. $B_1, \ldots, B_n$, called the *body*, is a sequence of FO atomic formulas over $\mathcal{V}$, Δ_{spec}, and $\mathcal{S} \cup \mathcal{S}^{pre} \cup I \cup \mathcal{C}^{rcv}$,
3. and $c_1, \ldots, c_m$ is a sequence of inequalities $t_1 \neq t_2$ where $t_1, t_2 \in \Delta_{spec} \cup \mathcal{V}$.

The rule is *safe* if each variable appearing in the head or in some inequality appears also in the body. A *posD2C program* over Λ is a finite set of posD2C rules over Λ.

Notation 1. Given FO formulas $B_1, \ldots, B_n$ over $\mathcal{V}$, Δ_{spec}, and $\mathcal{S}$, we write **prev** $\{B_1, \ldots, B_n\}$ to denote $B_1^{pre}, \ldots, B_n^{pre}$. Moreover, we denote the set of rules $\{H_1$ **if** $B., \ldots, H_n$ **if** $B.\}$ by '$\ H_1; \ldots; H_n$ **if** B.'.

Since posD2C programs are special forms of Datalog programs, they inherit the Datalog semantics. The semantics of D2C is given as in Datalog, with the provision that D2C works over extensional DBs of a specialized form: it contains information about the previous state, the incoming input, and a single incoming message (i.e., a single tuple over the signature $\mathcal{C}^{rcv}$).

Definition 9. Given a posD2C $\mathcal{P}$ and DBs D over $\mathcal{S}$, I over I, and $\{m\}$ over $\mathcal{C}^{rcv}$, the *output* $\mathcal{P}(D, I, m)$ of $\mathcal{P}$ over D, I, and m is the DB computed according to Datalog semantics by $\mathcal{P} \cup D \cup I \cup \{m\}$).

In what follows, we are not interested in the D2C output as a whole, but as split into a state and a channel part, used as new previous state and new messages on the channel. Specifically, we denote: by state($\mathcal{P}, D, I, m$) is the projection H_S of $\mathcal{P}(D, I, m)$ over $\mathcal{S}$; by channel($\mathcal{P}, D, I, m$) the DB H_C^{plain} where H_C is the projection of $\mathcal{P}(D, I, m)$ over $\mathcal{C}^{snd}$. Since Datalog with inequalities is monotone, also posD2C programs are monotone:

Lemma 1. *Given a posD2C program $\mathcal{P}$ and DBs D_1 and D_2 over $\mathcal{S}^{pre}$, I_1 and I_2 over $\mathcal{I}$, and $\{m\}$ over $\mathcal{C}^{rcv}$ such that $D_1 \subseteq D_2$, and $I_1 \subseteq I_2$, it is true that $\mathcal{P}(D_1, I_1, \{m\}) \subseteq \mathcal{P}(D_2, I_2, \{m\})$, state($\mathcal{P}, D_1, I_1, \{m\}$) $\subseteq$ state($\mathcal{P}, D_2, I_2, \{m\}$), and channel($\mathcal{P}, D_1, I_1, \{m\}$) $\subseteq$ channel($\mathcal{P}, D_2, I_2, \{m\}$).*

It is immediate to see that D2C programs inherit also genericity of Datalog.

Definition 10. Given a finite set $\Delta_{spec} \subseteq \Delta$, a Δ_{spec}-*isomorphism* is a bijection $\varphi : \Delta \longrightarrow \Delta$ such that, for each $c \in \Delta_{spec}$, $\varphi(c) = c$. We say that a DB D_1 is embedded in a DB D_2 via a Δ_{spec}-isomorphism φ, denoted $D_1 \sqsubseteq_\varphi D_2$, if $\varphi(D_1) \subseteq D_2$. We write $D_1 \sqsubseteq D_2$ if there is some φ such that $D_1 \sqsubseteq_\varphi D_2$.

Lemma 2. *Given a posD2C program $\mathcal{P}$, a Δ_{spec}-isomorphism, DBs D_1 and D_2 over $\mathcal{S}^{pre}$, I_1 and I_2 over $\mathcal{I}$, and $\{m\}$ over $\mathcal{C}^{rcv}$ such that $D_1 \sqsubseteq_\varphi D_2$, and $I_1 \sqsubseteq_\varphi I_2$, it is true that $\mathcal{P}(D_1, I_1, \{m\}) \sqsubseteq_\varphi \mathcal{P}(D_2, I_2\{m\})$, state$(\mathcal{P}, D_1, I_1, \{m\}) \sqsubseteq_\varphi$ state$(\mathcal{P}, D_2, I_2, \{m\})$, and channel$(\mathcal{P}, D_1, I_1, \{m\}) \sqsubseteq_\varphi$ channel$(\mathcal{P}, D_2, I_2, \{m\})$.*

3.2. PosIDP definition

Syntactically, a posIDP is a computational node with a self-loop channel running a posD2C program where the only possible message is $start$. This captures the case where the messages are used only as a triggering mechanism for the node computation, instead of a proper communication mechanism.

Definition 11. A posIDP $\mathcal{D}$ is a tuple $\mathcal{D} = \langle \Lambda, \mathcal{P}, D_0 \rangle$ where $\Lambda = \langle \Delta_{spec}, \mathcal{S}, \mathcal{I}, \{start\} \rangle$, $\mathcal{P}$ is a posD2C program over Λ, and D_0, called *initial state-DB*, is a DB over $\mathcal{S}$ with $\Delta(\mathcal{S}) \subseteq \Delta_{spec}$.

posIDP semantics is given in terms of configuration graphs. Different input policies (bounded and unbounded) amount to different graphs.

Definition 12. A configuration C of a posIDP $\mathcal{D} = \langle \Lambda, \mathcal{P}, D_0 \rangle$ is a pair $C = \langle D, M \rangle$ where D, called *state-DB*, is a DB over $\mathcal{S}$ and M is either $\{start\}$ or $\emptyset$. We denote the set of configurations of $\mathcal{D}$ by $\text{Conf}_{\mathcal{D}}$. The initial configuration of $\mathcal{D}$ is $\langle D_0, \{\text{start}\} \rangle$.

The unbounded input semantics is obtained by feeding the program, at each step, with an arbitrary input DB next to a message coming from the channel. If the channel is empty, the computation stops.

Definition 13. The input-unbounded configuration graph of a posIDP $\mathcal{D} = \langle \Lambda, \mathcal{P}, D_0 \rangle$ is the directed graph $\Upsilon = \langle \text{Conf}_{\mathcal{D}}, \rightarrow \rangle$ such that $(D_1, M_1) \rightarrow (D_2, M_2)$ if $M_1 = \{start\}$ and there is a DB I over $\mathcal{I}$ such that $D_2 = \text{state}(\mathcal{P}, D_1^{pre}, I, \{start^{rcv}\})^{plain}$ and $M_2 = \text{channel}(\mathcal{P}, D_1^{pre}, I, \{start^{rcv}\})^{plain}$

For $n \in \mathbb{N}$, the n-input bounded semantics is obtained by restricting input DBs containing at most n constants.

Definition 14. For $n \in \mathbb{N}$, the n-input-bounded configuration graph of a posIDP $\mathcal{D} = \langle \Lambda, \mathcal{P}, D_0 \rangle$ is the directed graph $\Upsilon_n = \langle \text{Conf}_{\mathcal{D}}, \rightarrow \rangle$ defined as in Def. 13 with the provision that I is a DB over $\mathcal{I}$ such $|\Delta(I)| \leq n$.

Note that even under input-bounded semantics, while an infinite amount of data may still flow through the system.

3.3. PosIDP n-coverability

In this paper, we study the decidability of n-coverability of posIDP. Technically, our study will focus on a simple target of the form $\{q_f^{pre}\}$, where q_f^{pre} is propositional.

Definition 15. Given $n < \omega$ (respectively, $n = \omega$), the n-*coverability problem* for posIDP asks, given a posIDP $\mathcal{D}$ and a *coverability target* DB T over $\mathcal{S}$, whether there is a path in Υ_n (Υ) from the initial configuration to a configuration $\langle D, M \rangle$ such that $D \subseteq T$.

These problems are interesting because, while arguably simple and classic, their variants over non-positive single-node CDPs are undecidable.[2] Thus, they are an excellent case study to check the impact of positive programs on the decidability of CDP verification. For example, undecidability of coverability problems for posIDP immediately returns undecidability for any model checking problem over the same or more expressive posIDP or CDP variant (e.g., with arbitrary node networks) based on languages capable of expressing coverability, such as the CDP-CTL fragments from [19]. Moreover, the next example shows that posIDP coverability can be used to check plan existence in some (simple, for the sake of brevity) planning domain.

[2]This is an immediate consequence of the proofs in [15].

```
 1 %Making facts persistent.
 2 succ(X,Y) if succ^{pre}(X,Y).
 3 block(X) if block^{pre}(X,Y).
 4
 5 %Making the posIDP non-terminating.
 6 start^{snd} if start^{rcv}.
 7
 8 %Alternating phases while updating the
       step id.
 9 phase(put) if prev{phase(pick), step(X)}.
10 phase(pick) if prev{phase(put), step(X)}.
11 step(X) if prev{step(X), phase(pick)}.
12 step(Y) if prev{step(X), succ(X,Y),
       phase(put)}.
13
14 %Using the input to pick blocks.
15 picked(X) if select(X), block(X),
       prev{free(X), phase(pick)}.
16 free(Y) if picked(X), free^{pre}(Y), X≠Y.
17 free(Y) if picked(X), on^{pre}(X,Y), Y≠table.
18 on(Y,Z) if picked(X), prev{on(Y,Z),
       phase(pick)}, X≠Y.
19
20 %Using the input to put blocks.
21 picked(X) if prev{picked(X), phase(put)}.
22 putOn(X) if select(X), block(X),
       prev{free(X), phase(put)}.
23 putOn(table) if select(table),
       phase^{pre}(put).
24 on(X,Y) if picked(X), putOn(Y).
25 on(X,Y) if prev{on(X,Y), phase(put)}.
26 free(X) if picked(X), phase^{pre}(put).
27 free(Z) if on(X,Y), free^{pre}(Z), Y≠Z,
       Z≠table.
```

(a)

```
28 %Encode number n and
       initialize step id.
29 succ(num_0,num_1).
30 ⋮
31 succ(num_{n-1},num_n).
32
```

```
33 %Encode initial block
       configuration.
34 block(a).
35 block(b).
36 block(c).
37 on(a,table).
38 on(b,table).
39 on(c,table).
```

```
40 free(a).
41 free(b).
42 free(c).
43
44 %Initialize step and phase.
45 step(num_0).
46 phase(pick).
```

(b)

Figure 1: (a) Program and (b) initial state-DB of the posIDP in Ex. 1. Lines starting with % are in-line comments.

Example 1. Consider a *Block World* problem where a robot has to stack the blocks a, b, and c, so that a sits on top of b which, turn, sits on top of c. The blocks are initially on a table. At each step the following actions can be performed: a block on a table or on top of a stack can be picked; the picked block can be put on the table or stacked on top of another block. At each moment, only one block can be picked.

For each $n \in \mathbb{N}$, we exhibit an instance of interactive 1-coverability that returns true if and only if a plan of at most n actions for the Block World problem above exists. We work with a posIDP $\mathcal{D}_n$ which non-deterministically attempts the computation of a plan. The program of $\mathcal{D}$ is in Fig. 1a while the initial DB is in Fig. 1b. Note that the program makes no use of negation (beyond inequalities) and only rule 6 sends messages, specifically a single $start$ message at each step after the consumption of the previous one. Thus, $\mathcal{D}$ is posIDP.

The signature of $\mathcal{D}$ is $(\Delta_{spec}, \mathcal{S}, \mathcal{I}, \mathcal{C})$ where Δ_{spec} is the set of constants appearing in the program or in the initial state-DB, $\mathcal{I} = \{select/1\}$, $\mathcal{C} = \{start\}$, and $\mathcal{S}$ is the set of all other relational symbols appearing in the program or initial state-DB. The number n is encoded in the initial state DB as a path from the constant num_0 to the constant num_n through constants num_i for $i \in \mathbb{N}$ via the binary relation $succ/2 \in \mathcal{S}$. This encoding is made persistent by rule 2 in the program. The program alternates two phases: a *pick phase* and a *put phase*, signaled respectively by the facts $phase(pick)$ and $phase(put)$. A pair of consecutive pick and put phases constitutes a step. During the pick phases, lines $15 - 18$ use the single constant, if any, in the input DB to select a free block, i.e., a block with no block on top of it. If the input provides a constant that does not match a block name, then no $phase(put)$ fact is deduced, stopping the non-deterministic computation of the plan. Similarly, during put phases, the program uses the input to select another free block, on which the previously picked block is stacked on. Again, if the input does not indicate a free block, the next phase does not happen. During these phases, the status of the blocks (whether they are free and where they sit) is encoded in the predicates *free*/1 and $on/2$ of $\mathcal{S}$. When the put phase of the n-th step terminates, since there is no successor of the constant num_n according to the extension of $succ$, no $phase(pick)$ fact is deduced, i.e., the next phase does not happen.

A plan exists if $\mathcal{D}$ covers the target $\{on(a, b), on(b, c), on(c, table)\}$. Moreover, we could have used the simpler target $\{exists\}$ if we have added the rule '$exists$ **if** $on(a, b), on(b, c), on(c, table)$.' If desired, upon coverability, one can also obtain the computed plan by inspecting the extension of the additional relation $action/3 \in \mathcal{S}$ whose dynamics is given by the following extra rules:

```
47 action(X,Y,Z) if action^pre(X,Y,Z).
48 action(pick, X, Y) if picked(X), prev{step(Y)}.
49 action(put, X, Y) if put(X), prev{step(Y)}.
```

Note that, since rule 6 makes the posIDP non-terminating, at the cost of losing track of the actions, we can check existence of plans of arbitrary length by modifying the program by dropping the rules $11, 12, 47-49$ and by removing the *step*-atoms from the bodies of rules 9 and 10. Alternatively, we can employ a strategy to initialize the encoding of an arbitrary natural number as in Sec. 4.1 below.

4. Undecidability

We now discuss undecidability of coverability for posIDP. We first consider 1-coverability, then n-coverability for each finite $n > 0$, and finally ω-coverability. In each case, we provide a reduction from deterministic 2CM termination, which is undecidable. This is done in two steps. First, we show that the simulation of deterministic 2CM whose counters are known to be bounded by a natural number n can be performed (even without using the input DB) by taking advantage of an encoding of natural numbers similar to the one in Ex. 1. Second, we exhibit a posIDP that initializes a non-deterministically chosen number n.

4.1. 1-Coverability

Definition 16. Given a $k \leq \omega$, a 2CM K is a k-2CM if k is the minimum ordinal such that, for each configuration $\langle q, n_1, n_2\rangle$ reachable from the initial configuration of K, $n_1 \leq k$ and $n_2 \leq k$.

Definition 17. Given a 2CM configuration $C = \langle q, n_0, n_1\rangle$ and a $n \geq \max\{n_0, n_1\}$, the n-*relational encoding* of C is the following DB, denoted by $[C]$:

```
1 succ(num_0,num_1).
2 ⋮
3 succ(num_{m-1},num_n).
```

```
4 min(num_0).
5 q.
6 c_0(num_{n_0}).
7 c_1(num_{n_1}).
```

In Def. 17, facts $1-3$ encode the numbers $0, \ldots, n$ using constants $num_0, \ldots, num_n$. Fact 4 highlights that num_0 behaves as zero. Facts $5-7$ encode that the current state is q and that the value of counter c_0 (c_1) is n_0 (n_1).

In what follows, we work with a fixed 2CM $K = \langle Q, \Delta, q_{init}, q_{fin}\rangle$. Given an $n \in \mathbb{N}$, we define a posIDP $IDP_n(K) = \langle \Lambda, D_0, \mathcal{P}\rangle$. The signature is $\Lambda = (\Delta_{spec}, \mathcal{S}, \mathcal{I}, \mathcal{C})$ where $\Delta_{spec} = \{num_0, \ldots, num_n\}$, $\mathcal{S} = \{q/0 \mid q \in Q\} \cup \{min/1, succ/2, c_0/1, c_1/1\}$, $\mathcal{I} = \emptyset$, and $\mathcal{C} = \{start/0\}$. The initial state DB D_0 is the n-relational encoding of the initial configuration of K. The program $\mathcal{P}$ contains the following rules:

```
1 start^snd if start^rcv.
2 succ(X,Y) if succ^pre(X,Y).
3 min(X) if min(X)^pre.
```

and, for each increment instruction $\langle q, +, i, q'\rangle$:

```
4 c_{i-1}(X) if prev{c_{1-i}(X), q}.
5 c_i(Y) if prev{c_i(X), succ(X,Y), q}.
6 q' if prev{c_i(X), succ(X,Y), q}.
```

and, for each decrement instruction $\langle q, -, i, q', q'' \rangle$:

```
7 c_{i-1}(X) if prev{c_{1-i}(X), q}.
8 c_i(Y) if prev{c_i(X), succ(Y,X), q}.
9 q' if prev{c_i(X), succ(Y,X), q}.
```

```
10 c_i(X) if prev{c_i(X), min(X), q}.
11 q'' if prev{c_i(X), min(X), q}.
```

Note that n does not affect the program, but only the initial state-DB. Rule 1 maintains the channel configuration, while rules 2 and 3 make the extension of *succ* and *min* persistent. At each step, the program simulates a 2CM transition δ. Rules 4 and 7 make sure that the encoding of the counter not involved in δ does not change. Rules 6 and 8 perform the increment and the decrement, if possible, i.e., if the current value of the updated counter has a successor or predecessor according to *succ*. Rule 10 makes sure that, if the decrement is not possible, i.e., the counter involved in δ is set to 0, then it is not changed. Rules 6, 9, and 11 update the state according to δ. Moreover, $IDP_n(K)$ is deterministic, i.e., each configuration has at most a unique successor in the configuration graph. Thus, $IDP_n(K)$ exhibits a single maximal computation run. The next lemmas capture the intuition that $IDP_n(K)$ simulates K along runs where the counters do not exceed n. If the counters exceed n, then the encoding breaks and the final state cannot be covered.

Lemma 3. *Let $n \in \mathbb{N}$. If K exhibits a run $\langle q_0, 0, 0 \rangle = C_0 \rightarrow^{\delta_1} C_1 \rightarrow^{\delta_2} \cdots \rightarrow^{\delta_k} C_k$ where, for each $i \leq k$, $C_i = \langle q_i, n_0^i, n_1^i \rangle$ and $n_0^i, n_1^i \leq n$, then $IDP_n(K)$ exhibits a run $\langle [C_0], \{start\} \rangle \rightarrow \langle [C_1], \{start\} \rangle \rightarrow \cdots \rightarrow \langle [C_k], \{start\} \rangle$.*

Proof. By induction on k. If $k = 0$, then the statement holds because the initial state-DB of $IDP_n(K)$ is $[C_0]$. Let the statement hold for some $k \in \mathbb{N}$, and let K exhibit a run $C_0 \rightarrow^{\delta_1} \cdots \rightarrow^{\delta_{k+1}} C_{k+1}$ as in the statement. By induction, $IDP_n(K)$ exhibits a non-maximal run $\langle [C_0], \{start\} \rangle \rightarrow \cdots \rightarrow \langle [C_k], \{start\} \rangle$.

We assume, without loss of generality, that δ_{k+1} operates on counter 0. If δ_{k+1} is an increment instruction, then it has to be of the form $\langle q_i, +, 0, q \rangle$ for some $q \in Q$. Thus $n \geq n_0^{k+1} = n_0^k + 1$. Hence, $succ(num_{n_k}, num_{k+1}) \in [C_k]$. Consequently, There is only one instantiation that satisfies the bodies of the rules of type $4-6$ for δ_{k+1}: the one binding X with num_{n_k} and Y with $num_{n_{k+1}}$. Moreover, for each instruction in Δ, there is no binding that satisfies the bodies of rules $7-11$. Thus, from $\langle [C_k], \{start\} \rangle$, the reception of the message *start* results in the configuration $\langle [C_{k+1}], \{start\} \rangle$.

If δ is a decrement instruction then the proof is analogous. □

Lemma 4. *If $IDP_n(K)$ exhibits a run $\langle D_0, \{start\} \rangle \rightarrow \cdots \rightarrow \langle D_i, \{start\} \rangle \rightarrow \cdots \rightarrow \langle D_k, \{start\} \rangle$ for some $i, k \in \mathbb{N}$ and $D_i \cap Q = \emptyset$, then, for each $j \geq i$, $D_j \cap Q = \emptyset$.*[3]

Proof. Each rule with some $q' \in Q$ in the head also has a q^{pre} in the body, for some $q \in Q$. □

The following lemmas are easy consequences of the previous.

Lemma 5. *If K is an ω-2CM, then K does not terminate and $IDP_n(K)$ does not 1-cover $\{q_{fin}\}$.*

Lemma 6. *If K is a k-2CM for some $k \in \mathbb{N}$, then, for each $n < k$, $IDP_n(K)$ does not 1-cover $\{q_{fin}\}$.*

Lemma 7. *If K is a k-2CM for some $k \in \mathbb{N}$, then, for each $n \geq k$, $IDP_n(K)$ 1-covers $\{q_{fin}\}$ if and only if K terminates.*

[3]Recall that Q is the set of states of the 2CM K.

We now show that there is a 1-bounded posIDP $IDP_{init}(K)$ such that, for each $n \in \mathbb{N}$, there is a finite maximal run that computes the initial state-DB of $IDP_n(K)$. If a maximal run does not initialize a $IDP_n(K)$, then it is infinite. Moreover, when we extend the program of $IDP_{init}(K)$ with that of $IDP_n(K)$, we obtain a posIDP $IDP(K)$ that covers $\{q_{fin}\}$ if and only if K terminates. The initial state-DB of $IDP_{init}(K)$ is $\{charge, min(num_0)\}$. The program unfolds two phases, called charge and generate. the first one charges a unary relation num. Each phase finishes when the input DB is $\{I(next)\}$.

```
1 min(X) if min^pre(X).
2 num(X) if prev{num(X), charge}.
3 charge; num(X) if I(X), charge^pre, X≠next.
4 generate if I(next), charge^pre.
5 start^snd if start^rcv, charge^pre.
```

Rules 1 and 2 make min and num persistent during the phase. Rule 3 charges num and maintains the phase. Rule 4 skips to the next phase. Note that the charge phase may run forever. However, if it ends, it ends with the state DB $\{min(num_0), num(a_1), \ldots, num(a_n), generate\}$, for some constant $a_1, \ldots, a_n$ and $n \in \mathbb{N}$. Rule 5 makes the system reactive during this phase by sending a message on the channel at each step.

The second phase uses the input to pick and remove, one per step, constants from num. The picked constant is used as the next successor, starting from num_0. This ensures that the chain of successors does not contain repetitions. Again, the phase ends upon reception of $\{I(next)\}$. In that case, the initial state of K is deduced.

```
1 last(num_0) if generate, charge^pre.
2 succ(Y,X); last(X) if I(X), num(X),
      last(Y), generate^pre, X≠next.
3 succ(X,Y) if succ^pre(X,Y).
4 num(X) if I(Y), prev{generate, num(X)},
      X≠Y.
5 q_0 if I(next), generate^pre.
6 start^snd if start^rcv, last(X), generate^pre.
```

Rule 1 applies only when the phase shifts from charge to generate and marks num_0 as the last constant in the (currently empty) chain of successors. Rule 2 applies only when the input is not triggering another phase shift: if the input provides a constant currently in num, then the constant is added to the chain of successors and marked as the last one. Rule 3 makes $succ$ persistent. Rule 4 makes a part of the extension of num persistent. Specifically, only the constants not mentioned in the input are retained in num. This way, when rule 2 adds a constant to the extension of $succ$, rule 4 implicitly removes it from num. Rule 5 performs the phase shift. Rule 6 makes the system reactive up to the phase shift, as long as there is a fact involving $last$. In fact, because of $last(X)$ in rule 6, if the input does not provide a constant in num, then rule 2 gets inhibited, the extension of $last$ remains empty, rule 6 gets inhibited, and the computation stops.

Note that, at each step of the generate phase, either $IDP_{init}(K)$ terminates or it removes a constant from num. Since the extension of num is finite and no rule puts back constants in num, $IDP_{init}(K)$ always terminates. Summarising, the generate phase has the following two behaviors, where a configuration $\langle D, \{start\}\rangle$ of $IDP_{init}(K)$ is valid if D is isomorphic to the initial state DB of a $IDP_n(K)$, for some $n \in \mathbb{N}$:

1. If, during the computation, the input DB is always of the form $I(c)$ for some constant c currently in the extension of num, except for the last step, where the input DB is $\{I(next)\}$, then $IDP_{init}(K)$ terminates in a valid configuration.
2. If the empty input DB or an input DB providing a constant not currently in num is eventually received, then $IDP_{init}(K)$ terminates in a non-valid configuration.

Moreover, for each $n \in \mathbb{N}$ there is at least one run of $IDP_{init}(K)$ that terminates in a valid configuration.

Let $IDP(K)$ be the posIDP obtained by extending the program $\mathcal{P}$ of the $IDP_n(K)$ posIDP. If K is a k-2CM, $IDP(K)$ exhibits the following maximal runs:

1. Infinite runs never leaving the charge phase of $IDP_{init}(K)$.

2. finite runs that terminate during the generate phase of $\mathit{IDP}_{\mathit{init}}(K)$ in non-valid configurations.
3. Infinite runs that leave the generate phase of $\mathit{IDP}_{\mathit{init}}(K)$, on which Lemma 5 is applicable.
4. Finite runs that leave the generate phase of $\mathit{IDP}_{\mathit{init}}(K)$ initializing the initial state-DB of $\mathit{IDP}_n(K)$ for some $n < k$ (recall that K is a k-2CM), on which Lemma 6 is applicable.
5. Finite runs that leave the generate phase of $\mathit{IDP}_{\mathit{init}}(K)$ initializing the initial state-DB of $\mathit{IDP}_n(K)$ for some $n \geq k$, on which Lemma 7 is applicable.

Note that the runs of type $1-4$ never cover $\{q_{fin}\}$. Thus, taking into account the overall behaviour of $\mathit{IDP}(K)$, genericity of posD2C, Lemma 5, Lemma 6, and Lemma 7, we obtain the following theorem.

Theorem 2. *For each deterministic 2CM K, K terminates if and only if $\mathit{IDP}(K)$ 1-covers $\{q_{fin}\}$.*

Since deterministic 2CM termination is undecidable and $\mathit{IDP}(K)$ is a posIDP, we obtain the following result.

Corollary 1. *1-coverability of posIDP is undecidable.*

4.2. n-Coverability for Finite n

If $n \in \mathbb{N}$ and $n > 1$, the above construction does not immediately work for n-coverability. In fact, the reception of input DBs containing more than one constant may interfere with the generation of the chain of successors, resulting in constants with more than one immediate successor. However, as long as n is finite, we can fix the above construction so that:

1. If an input DB does not provide the constants $trash_1, \dots, trash_{n-1}$ plus another arbitrary constant, then no rule body is satisfied, resulting in an empty DB and terminating the computation.
2. Otherwise, all the $trash_i$ constants are ignored.

This can be achieved by:

1. adding an atom ok to all the rules in the program of $\mathit{IDP}(K)$,
2. for each rule using variables $X_1, \dots, X_m$, adding the inequalities $X_i \neq trash_j$, for $i \leq m$ and $j \leq n-1$,
3. adding the rule 'ok **if** $I(trash_1), \dots, I(trash_{n-1}), I(X), X \neq trash_1, \dots, X \neq trash_{n-1}$.'.

The resulting posIDP, interpreted under the n-input bounded semantics, behaves as a $\mathit{IDP}(K)$ interpreted under the 1-input bounded semantics, with the provision that the reception of the empty input DB results in termination with configuration $\langle \emptyset, \emptyset \rangle$. Thus, all the previous lemmas and theorems apply also for n-coverability, yielding the next generalization of Cor. 1

Corollary 2. *For each $n \in \mathbb{N}$, n-coverability of posIDP is undecidable.*

4.3. ω-Coverability

When we consider unbounded inputs, the solution in the previous section does not work. However, we can still show undecidability by modifying the way we encode numbers. Specifically, instead of using the relation $succ$ to induce a chain of successors, we use it to capture a less demanding partial order, as depicted in Fig. 2, and defined next.

Definition 18. For each $n \in \mathbb{N}$, a relational encoding $[n]$ of n is any DB defined inductively as follows:

1. if $n = 0$, then $[n] = \{min(num_0^0)\}$.
2. if $n > 0$, then $[n]$ extends $[n-1]$ with the following set of facts, for some $m \in \mathbb{N}$ and for each $i \in \mathbb{N}$ such that num_{n-1}^i is in a constant in $[n-1]$:

```
succ(num$_{n-1}^{i}$, num$_{n}^{0}$).
⋮
succ(num$_{n-1}^{i}$, num$_{n}^{m}$).
```

We can now update the encodings for 2CM configurations.

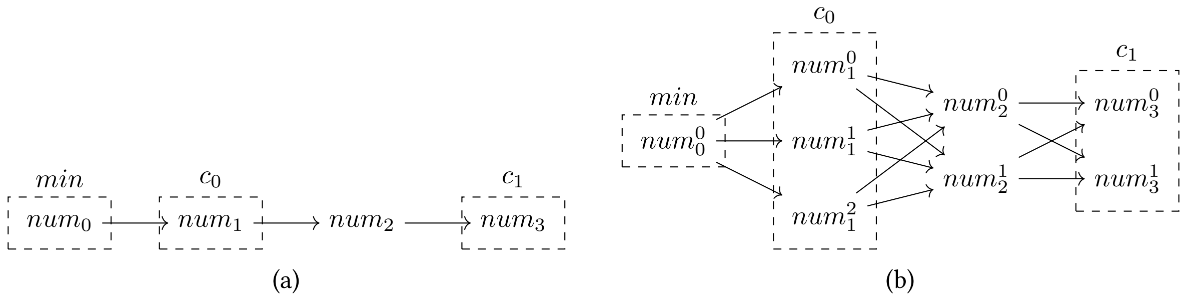

Figure 2: Depiction of (a) the 3-relational encoding, from Sec. 4.1, and (b) a [3]-relational encoding, from Sec. 4.3, of the 2CM configuration $\langle q, 1, 3\rangle$. The fact q is omitted. $succ(a, b)$ is represented by an arrow from a to b. Extensions of unary relations are depicted by dashed rectangles.

Definition 19. Given $n \in \mathbb{N}$, a relational encoding $[n]$ of n, and a configuration $C = \langle q, n_0, n_1\rangle$ where $n_0, n_1 \leq n$, the $[n]$-relational encoding of C is the DB $[n] \cup \{q\} \cup \{c_i(num^j_{n_i}) \mid i \in \{0, 1\}$ and $num^j_{n_i} \in \Delta([n])\}$, where $\Delta([n])$ is the active domain of $[n]$.

We now define $IDP_{[n]}(K)$ as $IDP_n(K)$, with the provision that the initial state DB is the $[n]$-relational encoding of the initial configuration of K. Note that Lemma 3 still applies with an analogous proof: instead of working with single facts $succ(num_k, num_{k+1})$, we just have to work with several facts $succ(num^i_k, num^j_{k+1})$. Lemma 4 immediately applies. Consequently, Lemma 5, Lemma 6, and Lemma 7 hold too. Interestingly, when $IDP_{init}(K)$ is exposed to unbounded input DBs, its behavior remains as in Sec. 4.1, with the provision that instead of initializing n-relational encodings, it initializes $[n]$-relational encodings. Specifically, during the charge phase, at each step all the constants in the input DB different from $next$ are internalized as part of num; the phase shift happens again when the input contains the constant $next$; during the generate phase, at each step, the input DB instructs the node to move several constants from num to $succ$, resulting in a $[n]$-relational encoding for some n. As before, only if the input DB does not contain any constant, the empty state DB is reached and the computation stops before initializing some $IDP_{[n]}(K)$.

Note that in this construction we did not modify the program or initial state-DB of $IDP(K)$. Thus, overall, we can use ω-coverability of $IDP(K)$ to check termination of K, even if adopting a different encoding. Thus, we obtain the following corollary.

Corollary 3. *For each $n \in \mathbb{N}$, n-coverability of posIDP is undecidable.*

5. DB-Minor Semantics

The undecidability of posIDP coverability, despite the monotonicity of posD2C, may appear surprising. However, the result is consistent with the insights provided by WSTS theory. In fact, the quasi-order used to define coverability, i.e., inclusion among DBs, is not a WQO among posIDP configurations. The next example provides a counter-example involving unary relations.

Example 2. The sequence $\{U(a_1)\}, \{U(a_2)\}, \{U(a_3)\}, \ldots$ for pairwise distinct constants $a_1, a_2, \ldots$ shows that inclusion of DBs over unary signatures is not a WQO. In fact, for each $i, j \in \mathbb{N}$, either $i = j$ or $\{U(a_i)\} \not\subseteq \{U(a_j)\}$.

An issue analogous to that in Ex. 2 affects also coverability problems in related models, such as data Petri Nets [5] and, more specifically, ν-Petri Nets [6], where each token carries a constant. This setting is reminiscent of DBs over at most unary signatures, where a token carrying a constant a in a place P amounts to the fact $P(a)$. In that setting, a WQO is obtained by considering inclusion of ν-Petri Net configurations up to isomorphisms. With this order, ν-Petri Nets are WSTSs. In the posIDP setting, this order amounts to inclusion of DBs over unary signatures up to isomorphisms. Unfortunately, even this order is not a WQO among arbitrary DBs, specifically when the signature includes binary relations, as showed in the next example.

Example 3. For each $n > 0$, let $C_n = \{E(a_0, a_1), \ldots, E(a_{n-1}, a_n)\}$ be a DB encoding a cyclic graph of length n, for some pair-wise distinct constants $a_1, \ldots, a_n$. The sequence $C_1, C_2, C_3, \ldots$ shows that inclusion up to isomorphisms of DBs over binary signatures is not a WQO. In fact, for each $i \in \mathbb{N}$, C_i is a cycle of length i, each isomorphic copy of C_i is a cycle of length i, but C_j does not contain any cycle of length j.

Ex. 3 highlights that, in order to apply WSTS theory to posIDP, we have to focus on coverability problems with respect to a quasi-order $\preceq$ among DBs such that $\preceq$ is WQO also over relational encodings of directed graphs. Thus, essentially, $\preceq$ has to capture a WQO among digraphs. Unfortunately, to the best of our knowledge, there are only few known WQOs among families of graphs and essentially only one among graphs in general, i.e., the graph-minor order [24]. Specifically, a graph G is a minor of a graph H if G can be obtained, up to a isomorphisms, from H by a sequence of vertex deletions, edge deletions, and edge contractions. The graph-minor relation can be used as a WQO also among directed graphs, when one forgets the direction of the edges, i.e., it works on their underlying graphs: given two directed graph G and H, G is a minor of H if the underlying graph of G is a minor of the underlying graph of H. When interpreting this order in the DB setting, the contraction of paths from vertex a to b amounts to the weak reachability of a constant b from constant a via a chain of pairs in a binary relation. Motivated by this insight, we propose the following quasi-order among DBs over at most binary signatures, which combines inclusion for propositions, inclusion up to isomorphisms for unary facts, and a contraction relation for binary facts.

Definition 20. Let $\mathcal{S}$ be a signature of at most binary symbols and φ a permutation of Δ. The $(\mathcal{S}, \varphi)$-*minor* relation $\preceq^{\varphi}_{\mathcal{S}}$ is the binary relation among DBs over $\mathcal{S}$ such that, for each two DBs D_1 and D_2 over $\mathcal{S}$, $D_1 \preceq^{\varphi}_{\mathcal{S}} D_2$ if and only if:

1. for each propositional symbol $P/0 \in \mathcal{S}$, if $P \in D_1$ then $P \in D_2$.
2. for each unary symbol $U/1 \in \mathcal{S}$ and constant a, if $U(a) \in D_1$, then $U(\varphi(a)) \in D_2$.
3. for each binary symbol $B/2 \in \mathcal{S}$ and two constants a and b, if $B(a, b) \in D_1$, then there is a sequence $c_0, \ldots, c_n$ of constants such that, $c_0 = \varphi(a)$ and $c_n = \varphi(b)$ and, for each $i < n$, $B(c_i, c_{i+1}) \in D_2$ or $B(c_{i+1}, c_i) \in D_2$.

The $\mathcal{S}$-*minor* $\preceq_{\mathcal{S}}$ is the union of all the $\preceq^{\varphi}_{\mathcal{S}}$ for all the permutations φ of Δ. The *DB-minor* relation $\preceq$ is the union of all the $\preceq_{\mathcal{S}}$ for all at most binary signatures $\mathcal{S}$.

In item 3 we require that c_i precedes c_{i+1} in D_2 or vice-versa. This is because we are capturing a WQO that works on top of the underlying graphs of the directed graphs encoded by the DBs. Because of the above arguments, we conjecture that $\preceq_{\mathcal{S}}$ is a WQO among DBs. However, irrespectively of the WQO status of $\preceq_{\mathcal{S}}$, the reductions in Sec. 4 apply also to coverability problems defined in terms of $\preceq$. In fact, in those reduction, we considered the coverability of propositional flags, on which $\subseteq$ and $\preceq$ coincide. Thus, WSTS theory cannot be applied to posIDP even for coverability problems based on $\preceq$ (otherwise decidability would apply). One reason for that is that posIDP are not compatible with $\preceq$. In other words, posIDP (and Datalog with inequalities) is monotone with respect to $\subseteq$ but not with respect to $\preceq$, as shown in the next example.

Example 4. Consider the one-rule program $\mathcal{P} = \{ok \ \textbf{if} \ \ B(X, Z), From(X), To(Z).\}$ and the DBs $D_1 = \{B(a, c), From(a), To(c)\}$ and $D_2 = \{B(a, b), B(b, c), From(a), To(c)\}$. While $D_1 \subseteq D_2$, $\mathcal{P}(D_1) \not\preceq \mathcal{P}(D_2)$.

Consequently, we argue that the only way to apply WSTS theory to posIDP requires not only to consider coverability based on $\preceq$, but also to modify the semantics of posD2C, and more precisely of Datalog, so as to avoid counter-examples such as Ex. 4. We call this semantic the *DB-minor Datalog semantics.* Technically, it coincides with the standard semantics of Datalog with the provision that a fact $R(a, b)$ is true in a DB D if $\{R(a, b)\} \preceq D$, instead of the traditional $R(a, b) \in D$ or, equivalently, $\{R(a, b)\} \subseteq D$. We call $\preceq$-posIDP the variant of posIDP defined on top of DB-minor Datalog semantics. We propose the following conjecture.

Conjecture 1. *The DB-minor Datalog semantics and $\preceq$-posIDP are well defined. Moreover, $\preceq$ is a WQO compatible with $\preceq$-posIDP and with effective pred-basis property, irrespectively of the bounds on the input DBs.*

By WSTS theory, a positive answer to the conjecture would immediately return decidability for $\preceq$-posIDP coverability problems based on $\preceq$.

6. Conclusions

We have studied the coverability problem for posIDP under bounded and unbounded input semantics. In all cases, we have obtained that coverability is undecidable. When compared to previous results [15, 18], these results indicate that the positivity of programs is irrelevant on the decidability of CDPs. We also investigated, from the perspective of WSTSs, why the monotonicity of Datalog is not enough to yield decidability for the restricted model of posIDP. The obtained insights motivated us to propose a semantics for Datalog and, more in general, posIDP and CDPs, based on the notion of graph-minor. This graph-minor semantics essentially closes each binary relation by reflexivity and transitivity.

As future works, we aim at proving Conjecture 1 and to study the actual expressiveness and practicality of Datalog-like languages and data-aware processes whose semantics is based on graph-minors or on other WQOs among DBs.

Declaration on Generative AI

The author has not employed any Generative AI tools.

References

[1] F. Di Cosmo, Decidability Boundaries in Formal Verification of Declarative Distributed Systems, Ph.D. thesis, Free University of Bozen-Bolzano, 2024.

[2] D. Brand, P. Zafiropulo, On communicating finite-state machines, J. ACM 30 (1983) 323–342. doi:10.1145/322374.322380.

[3] P. Chambart, P. Schnoebelen, Mixing lossy and perfect fifo channels, in: F. van Breugel, M. Chechik (Eds.), CONCUR 2008 - Concurrency Theory, 19th International Conference, CONCUR 2008, Toronto, Canada, August 19-22, 2008. Proceedings, volume 5201 of *Lecture Notes in Computer Science*, Springer, 2008, pp. 340–355. doi:10.1007/978-3-540-85361-9_28.

[4] P. A. Abdulla, C. Aiswarya, M. F. Atig, Data communicating processes with unreliable channels, in: M. Grohe, E. Koskinen, N. Shankar (Eds.), Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016, ACM, 2016, pp. 166–175. URL: https://doi.org/10.1145/2933575.2934535. doi:10.1145/2933575.2934535.

[5] R. Lazic, T. C. Newcomb, J. Ouaknine, A. W. Roscoe, J. Worrell, Nets with tokens which carry data, Fundam. Informaticae 88 (2008) 251–274.

[6] F. Rosa-Velardo, D. de Frutos-Escrig, Decidability and complexity of petri nets with unordered data, Theor. Comput. Sci. 412 (2011) 4439–4451. doi:10.1016/J.TCS.2011.05.007.

[7] F. Rosa-Velardo, Ordinal recursive complexity of unordered data nets, Inf. Comput. 254 (2017) 41–58. doi:10.1016/J.IC.2017.02.002.

[8] T. Roscoe, B. T. Loo, Declarative networking, in: L. Liu, M. T. Özsu (Eds.), Encyclopedia of Database Systems, Second Edition, Springer, 2018. doi:10.1007/978-1-4614-8265-9_1220.

[9] T. J. Ameloot, Declarative networking: Recent theoretical work on coordination, correctness, and declarative semantics, SIGMOD Rec. 43 (2014) 5–16. doi:10.1145/2694413.2694415.

[10] B. T. Loo, T. Condie, J. M. Hellerstein, P. Maniatis, T. Roscoe, I. Stoica, Implementing declarative overlays, in: A. Herbert, K. P. Birman (Eds.), Proceedings of the 20th ACM Symposium on

Operating Systems Principles 2005, SOSP 2005, Brighton, UK, October 23-26, 2005, ACM, 2005, pp. 75–90. doi:10.1145/1095810.1095818.

[11] J. Su, L. Wen, J. Yang, From data-centric business processes to enterprise process frameworks, in: S. Hallé, R. Villemaire, R. Lagerström (Eds.), 21st IEEE International Enterprise Distributed Object Computing Conference, EDOC 2017, Quebec City, QC, Canada, October 10-13, 2017, IEEE Computer Society, 2017, pp. 1–9. doi:10.1109/EDOC.2017.11.

[12] M. Gebser, R. Kaminski, B. Kaufmann, T. Schaub, Multi-shot ASP solving with clingo, Theory Pract. Log. Program. 19 (2019) 27–82. doi:10.1017/S1471068418000054.

[13] P. Cabalar, R. Kaminski, P. Morkisch, T. Schaub, telingo = ASP + time, in: M. Balduccini, Y. Lierler, S. Woltran (Eds.), Logic Programming and Nonmonotonic Reasoning - 15th International Conference, LPNMR 2019, Philadelphia, PA, USA, June 3-7, 2019, Proceedings, volume 11481 of *Lecture Notes in Computer Science*, Springer, 2019, pp. 256–269. doi:10.1007/978-3-030-20528-7_19.

[14] D. Calvanese, M. Montali, J. Lobo, Verification of fixed-topology declarative distributed systems with external data, in: D. Olteanu, B. Poblete (Eds.), Proceedings of the 12th Alberto Mendelzon International Workshop on Foundations of Data Management, Cali, Colombia, May 21-25, 2018, volume 2100 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2018.

[15] D. Calvanese, F. Di Cosmo, J. Lobo, M. Montali, Convergence verification of declarative distributed systems, in: S. Monica, F. Bergenti (Eds.), Proceedings of the 36th Italian Conference on Computational Logic, Parma, Italy, September 7-9, 2021, volume 3002 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2021, pp. 62–76.

[16] B. B. Hariri, D. Calvanese, G. D. Giacomo, A. Deutsch, M. Montali, Verification of relational data-centric dynamic systems with external services, in: R. Hull, W. Fan (Eds.), Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2013, New York, NY, USA - June 22 - 27, 2013, ACM, 2013, pp. 163–174. doi:10.1145/2463664.2465221.

[17] F. Di Cosmo, Verification of prev-free communicating datalog programs, in: A. Dovier, A. Formisano (Eds.), Proceedings of the 38th Italian Conference on Computational Logic, Udine, Italy, June 21-23, 2023, volume 3428 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2023.

[18] F. D. Cosmo, Analyzing termination for prev-aware fragments of communicating datalog programs, in: A. Fensel, A. Ozaki, D. Roman, A. Soylu (Eds.), Rules and Reasoning - 7th International Joint Conference, RuleML+RR 2023, Oslo, Norway, September 18-20, 2023, Proceedings, volume 14244 of *Lecture Notes in Computer Science*, Springer, 2023, pp. 110–125. doi:10.1007/978-3-031-45072-3_8.

[19] C. Aiswarya, D. Calvanese, F. D. Cosmo, M. Montali, Verification of unary communicating datalog programs, Proc. ACM Manag. Data 2 (2024) 89. doi:10.1145/3651590.

[20] A. Finkel, P. Schnoebelen, Well-structured transition systems everywhere!, Theor. Comput. Sci. 256 (2001) 63–92. doi:10.1016/S0304-3975(00)00102-X.

[21] G. Higman, Ordering by divisibility in abstract algebras, Proceedings of the London Mathematical Society 3 (1952) 326–336.

[22] M. L. Minsky, Computation: Finite and Infinite Machines, Prentice-Hall, 1967.

[23] S. Abiteboul, R. Hull, V. Vianu, Foundations of Databases, Addison-Wesley, 1995. URL: http://webdam.inria.fr/Alice/.

[24] R. Diestel, Graph Minors, Springer Berlin Heidelberg, Berlin, Heidelberg, 2025, pp. 363–416.