Verification of Unary Communicating Datalog Programs

C. AISWARYA, Chennai Mathematical Institute, India
DIEGO CALVANESE, Free University of Bozen-Bolzano, Italy and Umeå University, Sweden
FRANCESCO DI COSMO and MARCO MONTALI, Free University of Bozen-Bolzano, Italy

We study verification of reachability properties over Communicating Datalog Programs (CDPs), which are networks of relational nodes connected through unordered channels and running Datalog-like computations. Each node manipulates a local state database (DB), depending on incoming messages and additional input DBs from external services. Decidability of verification for CDPs has so far been established only under boundedness assumptions on the state and channel sizes, showing at the same time undecidability of reachability for unbounded states with only two unary relations or unbounded channels with a single binary relation. The goal of this paper is to study the open case of CDPs with bounded states and unbounded channels, under the assumption that channels carry unary relations only. We discuss the significance of the resulting model and prove the decidability of verification of variants of reachability, captured in fragments of first-order CTL. We do so through a novel reduction to coverability problems in a class of high-level Petri Nets that manipulate unordered data identifiers. We study the tightness of our results, showing that minor generalizations of the considered reachability properties yield undecidability of verification, both for CDPs and the corresponding Petri Net model.

CCS Concepts: • **Theory of computation → Logic and verification**; **Modal and temporal logics**; **Verification by model checking**; *Distributed computing models*; • **Information systems** → *Relational database query languages.*

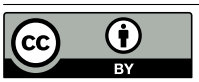
Additional Key Words and Phrases: Formal verification, Data-aware processes, Communicating Datalog Programs, Distributed computation, CTL-FO

ACM Reference Format:
C. Aiswarya, Diego Calvanese, Francesco Di Cosmo, and Marco Montali. 2024. Verification of Unary Communicating Datalog Programs. *Proc. ACM Manag. Data* 2, 2 (PODS), Article 89 (May 2024), 26 pages. https://doi.org/10.1145/3651590

1 INTRODUCTION

Declarative approaches to the specification of distributed data-aware systems have been extensively studied in the context of cloud computing [18], query computation over distributed databases [2, 3], software-defined networking and distributed protocols [21], as well as data access and exchange on the web [1]. These approaches share the general idea that the overall behavior of the system emerges from the interaction of a number of local components (hereafter called *nodes*), mutually connected in a given topology, each running a declarative program that describes at once the input/output behavior to exchange messages with the other nodes, and the update of the node internal state. Both the state and the exchanged messages are relational, thus making the overall system a distributed version of so-called *data-aware processes*, extensively studied within

Authors' addresses: C. Aiswarya, aiswarya@cmi.ac.in, Chennai Mathematical Institute, Chennai, India; Diego Calvanese, diego.calvanese@unibz.it, Free University of Bozen-Bolzano, 39100, Bolzano, Italy and Umeå University, 90187, Umeå, Sweden; Francesco Di Cosmo, francesco.dicosmo@unibz.it; Marco Montali, marco.montali@unibz.it, Free University of Bozen-Bolzano, 39100, Bolzano, Italy.



This work is licensed under a Creative Commons Attribution International 4.0 License.

© 2024 Copyright held by the owner/author(s).
ACM 2836-6573/2024/5-ART89
https://doi.org/10.1145/3651590

the foundations of data management from the modelling and static analysis point of view [9, 14, 26]. In [15, 16] systems of such form are studied, for which the communication takes place over queues, with different assumptions on their properties, e.g., presence or absence of lossiness. For such model, verification of LTL-FO and conversation protocols are studied, proving decidability and undecidability, with the former limited to cases of bounded queues.

In this work, we are interested in the static analysis of such distributed declarative data-aware processes, in the style of [9, 14, 26]. We focus in particular on the D2C language originally introduced in [21], which constitutes a prime representative of declarative approaches to distributed computation: it employs a suitably extended version of Datalog equipped with communication primitives and the possibility of referring to the previous and current node state.[1] On top of the resulting model of what we call *Communicating Datalog Programs (CDPs)*, two aspects become particularly important in the light of static analysis: *(i)* the presence of communication channels with different properties on faithfulness and ordering; *(ii)* the distinction between closed systems where new data are never created, but only the data present in the initial node states can be used and exchanged, and interactive systems where new data can be acquired and exchanged during the computation.

Declarative distributed systems with asynchronous communication occurring over multiset channels (where multiple copies of the same message may exist, even when the sender and receiver coincide) were considered in seminal works in the area, but only studied in connection with static analysis in [12]. The work in [12] considers interactive systems, reconstructing in a distributed setting the notions of external services and external user inputs widely studied for data-aware processes [9, 14, 26].

These two features make the resulting system infinite-state, with the consequence that even for very simple reachability properties, static analysis is undecidable [12]. Decidable subclasses have been singled out by importing and adapting the notion of *state-boundedness* originally introduced in [4, 5, 7], and applied in [4, 10, 13] to obtain decidability of verification of data-aware processes against rich variants of first-order branching-time temporal logics. In a state-bounded system, infinitely many objects may be seen within and across runs of the system, but in each single configuration reached during the computation, their number remains bounded. In the context of CDPs, this notion has a twofold effect: it essentially bounds the number of constants that can be simultaneously stored in each node state, as well as the size of each communication channel. Under such restrictions, it has been shown that model checking first-order CTL properties is decidable [12].

In this work, we start from the observation that bounding communication channels is a severe restriction, as it cannot be enforced even by suitably controlling how nodes are programmed, as nodes do not directly manipulate the content of communication channels, but only indirectly exploit them to exchange data. At the same time, [12] has shown that even propositional reachability is undecidable to check over severely restricted CDPs that employ messages with a binary signature. We consequently focus on the verification of *unary CDPs*, i.e., CDPs where the messages range over a signature of at most *unary* relational symbols, and while the local memory and interaction with external services is bounded, the channel capacity is *not*. This is also interesting to study in the light of multiset channels, since adopting queues, as in [15, 16], would immediately yield undecidability for unary, unbounded channels. We show that the resulting model is still powerful enough to model real-case scenarios, and engage in a fine-grained study of CDP verification against variants of reachability, expressed as fragments of first-order CTL. Specifically, we establish an

[1] For a full description of the language, its features, and how it relates to alternative proposals for declarative distributed computing, see [21].

equivalence between this problem and that of verifying coverability over a variant of Petri nets with unordered data [23], a property that is decidable to check despite the fact that these nets are essentially infinite-state. This yields decidability for positive nested reachability queries over unary CDPs, even in the case where the logic has not only the ability of querying the states, but also that of inspecting communication channels. We finally investigate the tightness of our decidability results, showing that minor generalizations fall back into undecidability.

2 PRELIMINARIES

2.1 Multisets

A (finite) multiset M over a set X is a function $X \longrightarrow \mathbb{N}$ such that the set $Supp(M) = \{x \mid M(x) > 0\}$, called *support*, is finite. The *set of multisets over* X is denoted by $X^{\oplus}$. For multisets M_1, M_2, we write $M_1 \leq M_2$ if $M_1(x) \leq M_2(x)$ for each $x \in Supp(M_1)$. Similarly, we define the sum $M_1 + M_2$ and the difference $M_1 - M_2$. The *size* $|M|$ of M is $\sum_{x \in Supp(M)} M(x)$. When needed, we extend a function $f : X \longrightarrow Y$ to $f : X^{\oplus} \longrightarrow Y^{\oplus}$, by setting $f(M)(y) = \sum_{\{x \in X \mid f(x)=y\}} M(x)$ for each $M \in X^{\oplus}$ and $y \in Y$. We denote multisets using set-like notation, with possibly repeated elements enclosed in the brackets $\{\!|$ and $|\!\}$. The empty multiset $\{\!|\}\!\}$ is also denoted by $\emptyset$.

2.2 ρ-Petri Nets

Unordered Data Petri Nets (UDPNs) [20] are extensions of Petri Nets[2] (PNs), in which tokens represent data. Here, we introduce them with a syntax (and corresponding semantics) convenient for our study. This form, called ρ-PN, is obtained by mixing the syntax of UDPNs in [23], in which tokens carry anonymous data, and that of ν-PNs [24], in which tokens carry explicit data. In ρ-PNs, data is represented by data identifiers, from a fixed infinite set ID containing a distinguished identifier $\bullet \in ID$.[3] We assume that $ID = \Delta \cup \{\bullet\}$, where Δ is the domain used to form DBs. To handle identifiers, transitions employ multisets of variables from $Var = \{v_i \mid i \in \mathbb{N}\}$.[4]

Definition 2.1. *A ρ-PN is a tuple (P, T, F), where (1) P and T are finite disjoint sets whose elements are called* places *and* transitions *respectively, (2) F, called* flow function*, maps pairs in $(P \times T) \cup (T \times P)$ to multisets of variables in $Var^{\oplus}$, and (3) for each $t \in T$, the set $Var(t) = \sum_{p \in P} Supp(F(p,t) + F(t,p))$ of* variables of t *is the initial segment $Var(t) = \{v_0, \ldots, v_{|Var(t)|-1}\}$ of Var.*

For each $t \in T$, we call pairs in $P \times \{t\}$ *pre-conditions for* t and pairs in $\{t\} \times P$ *post-conditions for* t. The flow function assigns variables to conditions, to be instantiated to identifiers moved by the transition. Moreover, for each $i \in \mathbb{N}$, the variable v_i is *special on* t if it does not occur on pre-conditions for t, i.e., for each $p \in P$, $v_i \in Var(t) \setminus Supp(F(p,t))$. Otherwise, i.e., there is some $p \in P$ s.t. $v_i \in Var(t) \cap Supp(F(p,t))$, v_i is *standard on* t. We denote by $spcVar(t)$ and $stdVar(t)$ the set of special and standard variables on t, respectively. We indicate special variables by the meta-symbol ρ and standard variables by the meta-symbols x, y, and z possibly with subscripts. Importantly, in denoting sets and multisets of variables, different meta-symbols denote different variables.[5]

The configurations of ρ-PNs are called *markings*, which, similarly to standard Petri Nets, assign a multiset of identifiers (intuitively carried by tokens) to each place. Given a marking, the flow function induces a set of enabled transitions. To enable a transition, variables on pre-conditions must be injectively instantiated, by a *mode*, to identifiers placed by the marking on the respective

[2]PNs are equivalent to Vector Addition Systems [19] restricted to non-negative integers.

[3]To distinguish ID from the identity function, we denote the latter by a lowercase id.

[4]For each $i \in \mathbb{N}$, we consider v_i as a variable, i.e., a string starting with the letter v and a subscript from $\mathbb{N}$, and not a meta-symbol used to denote variables. This is convenient to minimize the differences between ρ-PNs and UDPNs.

[5]E.g., the notation $\{x_1, x_2, y\}$ denotes a set of three distinct variables, instead of a non-empty set of at most three variables.

places. When an enabled transition fires, after instantiating also the post-conditions, the reached marking is computed according to the flow function.

Definition 2.2. *A* marking *for a ρ-PN $N = (P, T, F)$ is a function $P \longrightarrow ID^{\oplus}$. A* mode *for a transition $t \in T$ is an injective function $\sigma : Var(t) \longrightarrow ID$. The transition t is* enabled *by σ on a marking M for N if, for each $p \in P$, $\sigma(F(p,t)) \leq M(p)$. The marking M'* reached from *M by* firing *t enabled by σ on M, denoted by $M \rightarrow^{t,\sigma} M'$, is the marking s.t., for each $p \in P$,*

$$M'(p) = M(p) - \sigma(F(p,t)) + \sigma(F(t,p))$$

We write $M \rightarrow M'$ if $M \rightarrow^{t,\sigma} M'$ for some some t and σ. A marking M is reachable from a marking M_0 if there is a finite sequence of markings $M_0 \rightarrow M_1 \rightarrow \cdots \rightarrow M$.

Problem 2.3 (ρ-PN reachability).
Input: A ρ-PN N, an initial *marking M_0, and a* target *marking M.*
Output: Whether M is reachable in N from M_0.

To the best of our knowledge it is still unkown whether ρ-PN reachability is decidable [23]. A marking M' dominates a marking M, denoted $M \preceq M'$, if there is a permutation h of ID s.t. $h(\bullet) = \bullet$ and, for each place p, $h(M)(p) \leq M'(p)$.

Problem 2.4 (ρ-PN coverability).
Input: A ρ-PN N, an initial *marking M_0, and a* target *marking M.*
Output: Whether there is a marking M', reachable in N from M_0, s.t. $M \preceq M'$.

Theorem 2.5 ([23]). *The ρ-PN coverability problem is decidable, but with non-elementary complexity.*

ρ-PNs have a standard graphical representation (see, e.g., Fig. 3a). Places are depicted by circles, transitions by boxes, and each condition (c_1, c_2) by an arrow, from c_1 to c_2, labeled by $F(c_1, c_2)$. If $F(c_1, c_2) = \emptyset$, then the arrow is omitted. Markings are represented by writing, for each $p \in P$ and $d \in ID$, $M(p)(d)$ many d inside p.

3 SINGLE-NODE CDPs

In this section, we informally introduce the CDP model by Ma et al. [21]. However, for simplicity, we formalize only the fragment relevant for our study, which is the one over unordered channels, non-deterministic bounded inputs, and single-node networks.

3.1 The CDP Model

A CDP is a fixed network of data-centric nodes sharing messages via point-to-point channels. Each node (see Fig. 1) *(1)* runs a Datalog-like program, written in the language D2C, *(2)* updates its internal state, which is maintained as a *state DB* over a dedicated *state signature*, *(3)* receives information from the external environment, in the form of an *input DB* over a dedicated *input signature*, and *(4)* shares *messages*, i.e., single relational facts over a dedicated *transport signature*. The nodes react to incoming messages: when a message m from a node u is delivered to a node v, the latter gets activated and runs the program on its data-sources. In fact, the program input consists of the node state DB, the current input DB, the message m itself, and the local structure of the network at v, in the form of a *network DB*. The output provides a new state DB for v and a set of outgoing messages, each labeled by its recipient, which, in turn, are sent on the respective channels (without labels).

In this paper, we assume that communication is asynchronous and channels are reliable but unordered, that is, at each time-step, only one message is delivered (and, thus, only one node gets

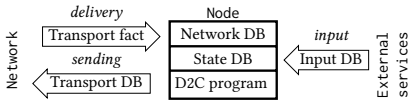


Fig. 1. A data-centric CDP node interacting in a network.

activated), no message can be lost, but the reception order is non-deterministic.[6] These assumptions are useful to model communication networks where message loss is ignored but order cannot be guaranteed (e.g., because of an underlying UDP transport protocol), or to represent processes (e.g., from a business scenario) where traveling resources (e.g., trucks with a cargo) are represented by messages. Note that several other choices are possible, e.g., synchronous communication over perfect or lossy channels (see [21]).

Since nodes react only to incoming messages, the communication network has, for each node, a self-loop channel (from the node to itself), which initially contains a special message dedicated to node activation. The self-loop channel can be used to model activation as a result of input DB updates, to encode CDPs over arbitrary networks of unordered channels to CDPs over single-node networks (lack of order is essential), and to abstract away parts of the network that, overall, react by sending messages back. Moreover, in single-node settings, the self-loop boils down to a form of auxiliary node memory, in the form of a multiset-DB from which information is non-deterministically extracted one tuple at a time.

CDP nodes are exposed to information from the external environment, which represents users and/or external services. Environment interaction is abstracted away by input policies, i.e., rules to provide a new input DB. In this paper, we focus on the b-bounded interactive input policies, where $b \in \mathbb{N}$: each time a node receives a message, the current input DB is substituted by a non-deterministically chosen new one with active domain of cardinality at most b. This policy is relevant to model interaction with external users which continuously provide new information, e.g., text messages for a chat application (see [12] for alternative policies).

All these information sources are manipulated by a D2C program, i.e., a set of Datalog-like rules specialized to the interactive and distributed setting of CDPs. The specialization is achieved by *(1)* organizing relation symbols in dedicated signatures (state, input, and transport), *(2)* using the in-rule flag `prev` to distinguish queries over the previous state DB and the new one under computation, and *(3)* labeling transport literals with terms representing senders or recipients (see [12] for a non-deterministic extension of D2C). However, in this paper, we focus on a simple D2C fragment, specialized for single-node networks (on which appended addresses are irrelevant, since there is only one address). In fact, while inconvenient for modelling, single-node CDPs are enough for the technical study of verification of CDPs employing unordered channels. Specifically, each such CDP can be encoded over a single node network.

At the cost of duplicating the signatures, it is possible to represent a set of state DBs via a single state DB. For example, the state DBs $S_1 = \{S(a)\}$, for node u, and $S_2 = \{S(b)\}$, for node v, both over the state signature $\mathcal{S} = \{S/1\}$, can be encoded in a single state DB $S = \{S^1(a), S^2(b)\}$, for node w, over a new state signature $\mathcal{S}^{\{1,2\}} = \{S^1/1, S^2/1\}$. The same applies also to unordered channels. In fact, by asynchronicity, a set of unordered channels, amounting to a set of multisets of messages, is analogous to their disjoint union (achieved by duplicating the transport signature), i.e., a single multiset capturing a single unordered channel (this does not happen for ordered

[6] Fairness is not imposed, since is often leads to undecidability [8], which would hide the impact of other assumptions. Lack of order is not equivalent to lossiness, since each message can be arbitrarily delayed only as long as other messages are on the channels. This is not the case when, e.g., the CDP approaches termination.

channels, amounting to queues). For example, the multisets $M_1 = \{m(a,b)\}$, of messages traveling from node u to node v, and multiset $M_2 = \{m(a,b), m(a,b)\}$, of messages traveling from node v to node u, both over the transport signature $\mathcal{T} = \{m/2\}$, are captured by the multiset $M = \{m^{u,v}(a,b), m^{v,u}(a,b), m^{v,u}(a,b)\}$, of messages traveling from node w to node w, on the transport signature $\mathcal{T} = \{m^{u,v}/2, m^{v,u}/2\}$. Moreover, also the program can be similarly modified so as to simulate, at node w, the asynchronous computation of a CDP with several nodes. Thus, without loss of generality, we focus on single-node networks only.

3.2 Single-node CDP Syntax

We now formalize the syntax of bounded-interactive, single-node CDPs. With a slight abuse, we refer to this fragment as, simply, CDPs and ignore all other CDP variants. When compared to full CDPs (see [12] for the full model), this fragment enjoys a streamlined syntax. For example, the single-node network topology is unique and, thus, left implicit. Moreover, since there is only one message destination, there is no need to make it explicit via labels on messages. The first syntactic component of CDPs are signatures.

Definition 3.1. *A* CDP signature *is a tuple $\Lambda = (\mathcal{S}, \mathcal{I}, \mathcal{T})$, where $\mathcal{S}$, $\mathcal{I}$, and $\mathcal{T}$ are pairwise disjoint relational signatures, respectively called* state, input, *and* transport. *A* state, input, *or* transport atom *(resp.,* literal*) is an atom (resp., literal) over $\mathcal{S}$, $\mathcal{I}$, or $\mathcal{T}$, respectively.*

CDP programs are, syntactically, reminiscent of stratified Datalog with negation and inequality constraints.

Definition 3.2. *A* D2C rule *over a CDP signature Λ is a formula*

$$H \text{ if } L_1, \ldots, L_n \text{ prev } L_{n+1}, \ldots, L_{n+m}, C_1, \ldots, C_h.$$

s.t.: (1) H is a state or transport literal, (2) $L_1, \ldots, L_n$ are state, input, or transport literals, (3) $L_{n+1}, \ldots, L_{n+m}$ are state literals, and (4) $C_1, \ldots, C_h$ are inequality constraints of the form $t_1 \neq t_2$, where t_1 and t_2 are terms (constants or variables). The rule head *is H and the rule* body *is $L_1, \ldots, L_n$ prev $L_{n+1}, \ldots, L_{n+m}, C_1, \ldots, C_h$. The rule* scope of `prev` *is $L_{n+1}, \ldots, L_{n+m}$. The rule is* safe *if each variable occurring in the head, in a negated literal, or in an inequality constraint, also occurs in a positive literal in the rule body. The rule is* transport consistent *if each variable occurring in a transport atom in the head also occurs in a positive non-input literal.*

Intuitively, in the scope of `prev`, state literals query the state DB available immediately before node activation. Outside the scope of `prev`, in the body, input literals query the input DB, transport literals the incoming message, and state literals the new state DB under computation. In the head, transport atoms deduce the outgoing messages and state atoms the facts in the new state DB. At the end of the computation, the new state DB substitutes the previous one and the outgoing messages are sent on the channel. Transport consistency states that data from the input DB cannot directly flow to the channel. This matches with the assumption that only nodes have the power to send messages, which have to be preliminarily gathered in an out-buffer that contributes to the node configuration (state DB, possibly affecting its boundedness, cf. Def. 3.9). Note that state literals in the scope of prev and transport literals in the body are not involved in forming recursive dependencies. While this feature appears as a major difference with Datalog, actually, it is just a matter of making the syntax convenient for the CDP semantics. In fact, this difference is ironed out by the Datalog encoding of rules. First, the encoding concerns CDP signatures, atoms, and literals.

Definition 3.3. *Given a CDP signature $\Lambda = (\mathcal{S}, \mathcal{I}, \mathcal{T})$, the Datalog encoding $\overline{\Lambda}$ of Λ is the signature $\mathcal{S} \cup \mathcal{S}' \cup \mathcal{I} \cup \mathcal{T} \cup \mathcal{T}'$ s.t. $\mathcal{S}' = \{S'/n \mid S/n \in \mathcal{S}\}, \mathcal{T}' = \{T'/n \mid T/n \in \mathcal{T}\}$, and $\mathcal{S}, \mathcal{S}', \mathcal{I}, \mathcal{T}$, and $\mathcal{T}'$ are pairwise disjoint. The signatures $\mathcal{S}'$ and $\mathcal{T}'$ are called* encoded state *and* encoded transport *signature.*

The encoding $\overline{A(t_1, \ldots, t_n)}$ of a state or transport atom $A(t_1, \ldots, t_n)$ is the atom $A'(t_1, \ldots, t_n)$. The encoding $\overline{A}$ of an input atom A is A itself. The encoding $\overline{\text{not } A}$ of a literal not A is the literal not $\overline{A}$.

D2C rules over Λ are encoded in Datalog rules over $\overline{\Lambda}$.

Definition 3.4. *Given a D2C rule π as in Def. 3.2, over a CDP signature Λ, the* Datalog encoding $\overline{\pi}$ of π *is the rule*

$$\overline{H} \text{ if } \mathcal{L}_1, \ldots, \mathcal{L}_n, L_{n+1}, \ldots, L_{n+m}, C_1, \ldots, C_h.$$

over $\overline{\Lambda}$, where, for each $i \in \{1, \ldots, n\}$, either $\mathcal{L}_i = \overline{L_i}$, if L_i is a state literal, or $\mathcal{L}_i = L_i$, if L_i is a transport literal. The Datalog encoding $\overline{\mathcal{P}}$ of a finite set $\mathcal{P}$ of D2C rules *over $\overline{\Lambda}$ is the set $\{\overline{\pi} \mid \pi \in \mathcal{P}\}$ of rules over $\overline{\Lambda}$. A* D2C program *$\mathcal{P}$ over a CDP signature Λ is a finite set of safe and transport consistent D2C rules s.t. $\overline{\mathcal{P}}$ is stratified.*

Syntactically, a CDP is a combination of signature Λ, program $\mathcal{P}$, initial state DB S_0, and initial message m_0 for self-loop.

Definition 3.5. *A* CDP *is a tuple $(\Lambda, \mathcal{P}, S_0, m_0)$, where Λ is CDP signature, $\mathcal{P}$ is a D2C program over Λ, S_0 is a state DB denoting the initial state, and m_0 is an initial message.*

3.3 Single-node CDP Semantics

The semantics of CDPs is given in terms of configuration graphs, which connect CDP configurations via transitions. Configurations describe snapshots of the system, including the state DB, the input DB, and the channel, represented as a multiset.

Definition 3.6. *Given a CDP $D = (\Lambda, \mathcal{P}, S_0, m_0)$, a* configuration *of D is a tuple (S, I, C), where S is a state DB, I is an input DB, and C is a multiset of facts over $\mathcal{T}$. A configuration (S, I, C) is* initial *if $S = S_0$ and $C = \{m_0\}$. The configuration is b-*input, *b-*state, or *b-*channel bounded, *for some $b \in \mathbb{N}$, if the cardinality of the active domain of I, of the active domain of S, or of C, respectively, is at most b.*

CDP transitions formalize computation steps, consisting of asynchronous message delivery, input update, program application, state update, and message sending. This includes D2C semantics, which is given via Datalog encodings (see Def. 3.4) and decodings.

Definition 3.7. *Given a CDP signature $\Lambda = (\mathcal{S}, \mathcal{I}, \mathcal{T})$, the* Datalog decoding *$\widetilde{S}$ of a DB S, over the encoded state or transport signature, is the DB $\{A(t_1, \ldots, t_n) \mid A'(t_1, \ldots, t_n) \in \mathcal{S}\}$. Given a DB S and a signature $\mathcal{S}'$, we denote by $S/\mathcal{S}'$ the projection of S over $\mathcal{S}'$, i.e., the DB $\{A(t_1, \ldots, t_n) \mid A(t_1, \ldots, t_n) \in S, A/n \in \mathcal{S}'\}$. Given a program $\mathcal{P}$, a state DB S, an input DB I, and a transport fact m, let O be the output of $\overline{\mathcal{P}}$ over the extensional DB $S \cup I \cup \{m\}$. The* evaluation *of $\mathcal{P}$ over the tuple (S, I, m) is the pair $(st(S, I, m), out(S, I, m))$, s.t. $st(S, I, m) = \widetilde{O}/\mathcal{S}$ and $out(S, I, m) = \widetilde{O}/\mathcal{T}$.*

We can now easily define CDP configuration graphs.

Definition 3.8. *Given a CDP $D = (\Lambda, \mathcal{P}, S_0, m_0)$ and $b \in \mathbb{N}$, the* b-bounded configuration graph *Υ^b of D is the directed graph $(V, \rightarrow)$ where V is the set of b-input bounded configurations of D and $C_1 \rightarrow C_2$ iff, for $i \in \{1, 2\}$, C_i is a configuration (S_i, I_i, C_i) of D, for some S_i, I_i, and C_i s.t. either (1) there is a message $m \in C_1$ for which $S_2 = st(S_1, I_1, m)$, and $C_2 = C_1 - \{m\} + out(S_1, I_1, m)$, or (2) $Supp(C_1) = \emptyset$ and $C_2 = C_1$.*[7]

Given $b \in \mathbb{N}$, we call b-CDP a CDP interpreted under Υ^b. CDPs may also be state and/or channel bounded. However, these are properties, enjoyed by some CDP, and not alternative semantics.

Definition 3.9. *Given $b, s, c \in \mathbb{N}$, a b-CDP is s-state or c-channel bounded if all configurations reachable in Υ^b from an initial configuration are s-state or c-channel bounded, respectively.*

```
1 st(D,ready) if addDrone(D) prev not
      st(D,ready), not st(D,duty).
2 tOff if launch(D), not land prev st(D,ready).
3 st(D,duty) if launch(D), tOff.
4 drone(D) if launch(D), not land prev
      st(D,ready).
5 launchFailed if not tOff, launch(D).
6 land if drone(D).
7 st(D,ready) if drone(D).
8 st(D,duty) if not drone(D) prev st(D,duty).
9 st(D,ready) if not launch(D) prev st(D,ready).
10 none if none.
```

Fig. 2. Program of the 1-CDP in Ex. 3.10.

Example 3.10. Fig. 2 contains the program $\mathcal{P}$ of a 1-CDP $D = (\Lambda, \mathcal{P}, \emptyset, \mathsf{none})$ that models a drone base. The signature is $\Lambda = (\mathcal{S}, \mathcal{I}, \mathcal{T})$ where $\mathcal{S} = \{\mathtt{st/2}, \mathtt{land/0}, \mathtt{launchFailed/0}, \mathtt{tOff/0}\}$, $\mathcal{I} = \{\mathtt{addDrone/1}, \mathtt{launch/1}\}$, $\mathcal{T} = \{\mathtt{none/0}, \mathtt{drone/1}\}$. External services can, at each step, register a drone (rule 1) or order a registered drone launch (rules 2, 3). The status of registered drones (`ready` or `duty`) is stored in the `st` state relation. Traveling drones are represented by `drone(d)` messages, for some drone name `d`, on the channel. Drone launches are simulated by sending the respective `drone` message (rule 4). Drone landings, one by one, are simulated by message reception (rules 6, 7). Drones can take off only if they are ready and no landing is occurring (rules 3, 4), otherwise a `launchFailed` flag is signaled (rule 5). The state of all other drones remains unmodified (rules 8, 9). To process inputs when no drone is landing, a message `none` is persistently maintained on the channel (rule 10). Initially, the channel contains one instance of the message `none`. Unordered channels naturally model the fact that drones can have different flight paths and, thus, land in non-deterministic order. D is neither state nor channel bounded (cf. Ex. A.1 in App. A).

4 PROBLEM

We study the problem of formal verification of CDPs. Previous work [12] showed that control-state reachability (that is, whether there is an initial configuration from which the target state DB is reachable — ignoring the configuration of the channel) is undecidable even for restricted CDPs that *(i)* have a single-node network, *(ii)* use the channel solely to (re)activate the node, and *(iii)* employ a unary state signature. Decidability can be gained by imposing boundedness conditions on the various CDP data sources [12]. In fact, for state- and channel-bounded CDPs, decidability holds for temporal model checking againts formulae in $\mathrm{CTL}_{\mathrm{CDP}}$, a branching-time logic that mixes CTL operators to analyze the system evolution, and FOL to query the data sources.

Unfortunately, boundedness is a semantic property, undecidable to check. In addition, while there are different techniques to enforce state boundedness [5, 25], the same does not hold for channels. Furthermore, as pointed out in the introduction, imposing boundedness is particularly restrictive for communication channels. Interestingly, while undecidability of control-state reachability over state-unbounded CDPs already holds for unary signatures, in the case of CDPs with bounded states and unbounded channels it has been proved only for binary transport signatures [12]. This makes CDPs that are state- and input-bounded, but operate over unbounded channels carrying unary messages, worth investigating. We call such CDPs *unary CDPs* (uCDPs).

In the following, we study the problem of model checking variants of uCDPs against selected fragments of $\mathrm{CTL}_{\mathrm{CDP}}$. The base-level fragment we use to express reachability-like properties called $\mathsf{EF}(-, bool, st)$, essentially, mixes EF CTL temporal operators with closed FO formulas over the state signature. Specifically, given a CDP $D = (\Lambda, \mathcal{P}, S_0, m_0)$, where $\Lambda = (\mathcal{S}, \mathcal{I}, \mathcal{T})$, the language

[7]This condition is necessary to make sure that each configuration has a successor.

$\mathsf{EF}(-, bool, st)$ over D is defined by the rules

$$\begin{array}{lcl} \Phi & ::= & \varphi \mid \mathsf{EF}\varphi \mid \Phi \wedge \Phi \mid \Phi \vee \Phi \\ \varphi & ::= & t_1 = t_2 \mid S(\mathbf{t}) \mid \exists x.\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \neg\varphi \end{array}$$

where Φ are temporal formulas, φ are closed FO formulas over $\mathcal{S}$, t_1 and t_2 are terms, and $\mathbf{t}$ is a tuple (of proper size) of terms. Such formulas are interpreted as follows: *(1)* $S(\mathbf{t})$ queries whether $S(\mathbf{t})$ is in the current state DB, *(2)* $\exists x$ is an existential quantifier over the active domain of the state DB and the support of the multiset channel, and *(3)* $\mathsf{EF}\varphi$ is interpreted as in standard CTL, i.e., there exists a path, in the CDP configuration graph, on which φ eventually holds [6]. For example, reachability of a state DB containing the fact $S(\mathsf{a})$ is expressed by the formula $\mathsf{EF}\ S(\mathsf{a})$, while reachability of the state that contains only that fact by $\mathsf{EF}\ S(\mathsf{a}) \wedge \neg\exists X.S(X) \wedge \neg X = \mathsf{a}$.

We study variants of reachability properties starting from $\mathsf{EF}(-, bool, st)$ and considering formulae consisting of (positive boolean combinations of) sentences starting with an EF operator, tuning them along three dimensions: which components they inspect (state DB or also channels)[8], the available temporal operators beyond EF, and the presence of negations in the FO queries. We identify such fragments with notation $\mathsf{EF}(\square, \triangle, \circ)$, where:

- $\square$ indicates which temporal operators can be nested; it can be one of: *(i)* $-$ (no nesting), as in the grammar above, *(ii)* EF^+ (nesting of multiple EF), obtained by adding rule $\Phi ::= \mathsf{EF}\Phi$ to the grammar, *(iii)* AG (nesting of a single AG), obtained by adding rule $\Phi ::= \mathsf{EF}(\varphi \wedge \mathsf{AG}\varphi)$, or *(iv)* AX^2 (nesting of two AX in the scope of a single EF), obtained by adding rule $\Phi ::= \mathsf{EF}(\varphi \wedge \mathsf{AXAX}\varphi)$;
- $\triangle$ indicates how negation is supported by FO formulas; it can be either *(i) bool*, as defined by the grammar above, or *(ii) pos* (no negation), obtained by dropping the rule $\varphi ::= \neg\varphi$;
- $\circ$ indicates whether formulas can only query state DBs, or also channels; it can be either: *(i) st* (queries only over node states), as defined by the grammar above, or *(ii) st+ch* (queries also over the support of channel multisets), obtained by adding rule $\varphi ::= T(\mathbf{t})$, where T is in the transport signature.

The formal syntax and semantics of these languages are defined in App. D. We study the model-checking problems of state bounded single-node CDPs against the $\mathsf{EF}(\square, \triangle, \circ)$ fragments for a given initial configuration.

Problem 4.1 ($\mathsf{EF}[\square, \triangle, \circ]$-MC). *Let* $\square \in \{-, \mathsf{EF}+, \mathsf{AG}, (\mathsf{EF}, \mathsf{AX})+\}$, $\triangle \in \{pos, bool\}$, *and* $\circ \in \{st, st{+}ch\}$.

Input: A $b \in \mathbb{N}$, $s \in \mathbb{N}$, *s-state bounded uCDP* D, *initial configuration* C_0, *and closed formula* $\Phi \in \mathsf{EF}(\square, \triangle, \circ)$.

Output: Whether the configuration graph Υ^b *satisfies* Φ *from* C_0.

Verification w.r.t. all initial configurations reduces to finitely many instances of $\mathsf{EF}[\square, \triangle, \circ]$-MC. Indeed, due to state and input boundedness, the initial configurations are finitely many up to isomorphisms, and FO formulas are invariant under isomorphisms that fix the constants in them. Establishing the decidability status of the different variants of this problem is challenging, due to the subtle interplay of the CDP components, e.g., how the node state is affected by the content of the multiset channel, whose access is limited by aysnchronous communication. To attack this problem, we provide a bridge with models and techniques for the verification of data-aware extensions of PNs, in particular ρ-PNs. PNs are one of the most widely studied models for concurrent computations, particularly suited to handle asynchronous threads and message passing. Specifically, ρ-PNs lend themselves to be connected to uCDPs. In fact, tokens carrying single data elements match constants used in unary messages, and places match unary relation symbols - so that inserting a token

[8]Input DBs are disregarded since the evolution of their extension is completely non-deterministic and its interaction with the state DB can be captured by slightly modifying the CDP program so as to include the bounded input DB in the state DB.

carrying constant c in a place M naturally corresponds to having message $\mathtt{M(c)}$ in the channel. What is not at all clear, instead, is how to encode in the ρ-PN the infinitely many input and state DBs that may be encountered along a computation. Recall, in fact, that even under state-boundedness, a CDP can encounter infinitely many, genuinely distinct state DBs.

To address this issue, we represent state and input DBs up to isomorphism. This can be done by introducing dedicated places for the following purposes: *(1)* to encode the relation symbols of messages; *(2)* to represent the isomorphism types of bounded input and state DBs, over a fixed representative bounded domain; *(3)* to specify a mapping from the representative domain to the infinite domain of data values used to form input and state DBs; *(4)* to deal with the special constants that are distinguished in the CDP program, ensuring that each one of those forms a singleton isomorphism type. This constitutes the basis for reducing $\mathsf{EF}[\square, \triangle, \circ]$-MC problems over uCDPs, to coverability checks over ρ-PNs.

We proceed as follows. We first investigate the decidability status of variants of control-state reachability for ρ-PNs (Sec. 5). We then transfer these results to uCDPs, showing reductions from variants of uCDP model checking to ρ-PNs control-state reachability (Sec. 6).

5 ρ-PN VERIFICATION

We introduce now the language P-ρCTL to express coverability properties on ρ-PNs and study the decidability of the related model-checking problem. P-ρCTL features the CTL EF temporal operator, boolean conjunctions and disjunctions, and replaces propositions with markings interpreted, each on its own, up to isomorphisms.

Definition 5.1. *Given a set P of places, P-ρCTL is the language of formulas φ defined by the following grammar:*

$$\varphi ::= M \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \mathsf{EF}\varphi$$

where M denotes a marking over the place set P. Markings M are called atomic P-ρCTL *formulas.*

The semantics of P-ρCTL is defined as for CTL, with the provision that, for a ρ-PN N and markings M and M', the current marking M of N satisfies an atomic formula M' if M covers, up to isomorphisms, M' (see App. B for the full semantics).

Problem 5.2 (P-ρCTL-MC).

Input: A ρ-PN $N = (P, T, F)$, marking M_0, and P-ρCTL formula φ.
Output: Whether N satisfies φ from M_0, denoted by $N, M_0 \models \varphi$.

Since atomic formulas perform coverability checks, P-ρCTL-MC can be reduced to plain ρ-PN coverability. This is done by induction on the structure of the P-ρCTL formula. First, a given formula φ, to be checked on a ρ-PN N and initial marking M_0, is represented as a syntax tree. Its leafs are the occurrences of atomic formulas and the other nodes are obtained by applying to the children the corresponding boolean or temporal operator.

Second, from leafs to the root, each node ψ is mapped to a ρ-PN N^ψ and initial marking M_0^ψ where *(1)* the net N^ψ contains, as sub-nets, the nets N^τ, for each sub-formula τ of ψ, *(2)* P^ψ contains the places check^ψ and cover^ψ, *(3)* M_0^ψ places at least a distinguished identifier on check^ψ, and *(4)* transitions are added so that the place cover^ψ can be marked with a distinguished identifier iff $N^\psi, M_0^\psi \models \psi$. For example, for the ρ-PN N and initial marking M_0 in Fig. 3a, and an atomic formula m_1, the net N^{m_1} in Fig. 3b is obtained by *(1)* dropping all transitions from N, *(2)* adding the places check^{m_1} and cover^{m_1}, *(3)* adding a transition $\mathit{checking}^{m_1}$ that *reads*[9] the marking m_1 (up to

[9] I.e., that consumes and puts back.

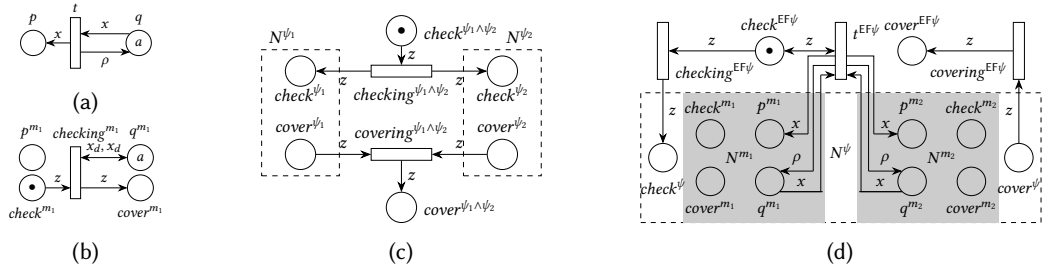


Fig. 3. (a) A ρ-PN N with initial marking M_0, (b) the ρ-PN N^{m_1}, with initial marking $M_0^{m_1}$, where $m_1(p) = \emptyset$, and $m_1(q) = \{d, d\}$, (c) a partial representation of the ρ-PN $N^{\psi_1 \wedge \psi_2}$ with initial marking $M_0^{\psi_1 \wedge \psi_2}$, and (d) a partial representation of the ρ-PN $N^{\mathsf{EF}\psi}$, with initial marking $M_0^{\mathsf{EF}\psi}$, where the atomic sub-formulas of ψ are m_1 and m_2. The dashed rectangles encapsulate partially represented sub-nets N^{ψ}, for the appropriate sub-formula ψ. The gray rectangles encapsulate the places of N^{m_i}, for $i \in \{1, 2\}$.

isomorphisms) and moves the distinguished identifier $\bullet$ from $check^{m_1}$ to $cover^{m_1}$, and *(4)* setting the initial marking $M_0^{m_1}$ to M_0 plus a $\bullet$ on $check^{m_1}$. Thus, the marking M_1, which places a $\bullet$ on $cover^{m_1}$, can be covered in N^{m_1} iff $checking^{m_1}$ can fire on $M_0^{m_1}$, i.e., iff M_0 covers m_1.

The constructions for non-leaves take into account the children nets and are depicted in Fig. 3c for conjunctions (disjunctions are analogous), and Fig. 3d for EF operators. The latter case is the most involved. Nodes $\mathsf{EF}\psi$ have only one child (i.e., ψ), but, possibly, several descendants related to the atomic sub-formulas m_i. The initial marking $M_0^{\mathsf{EF}\psi}$ places *(1)* for each leaf m_i, a copy of M_0 on the places of N^{m_i}, and *(2)* a $\bullet$ on $check^{\mathsf{EF}\psi}$. For each transition t of N, a transition $t^{\mathsf{EF}\psi}$ is added to $N^{\mathsf{EF}\psi}$, so that, as long as $check^{\mathsf{EF}\psi}$ is marked, $t^{\mathsf{EF}\psi}$ can fire, resulting in a coordinated update of the markings of all the sub-nets N^{m_i} of the leafs. Specifically, the sub-markings are updated as if the transition t of N fires with the same mode σ (induced by the *mode* enabling $t^{\mathsf{EF}\psi}$) in all sub-nets N^{m_i}, at the same time. Doing so, sub-nets of the leafs consistently represent the same marking of the net N reached after one firing of t under the single mode σ. This situation persists until another new transition, $checking^{\mathsf{EF}\psi}$, fires, moving the identifier from $check^{\mathsf{EF}\psi}$ to $check^{\psi}$, reaching the marking $M_1^{\mathsf{EF}\psi}$. At this point, the projection M_1^{ψ} of $M_1^{\mathsf{EF}\psi}$ to the net N^{ψ} behaves as an initial marking for N^{ψ}. By induction, the place $cover^{\psi}$ can be marked iff $N^{\psi}, M_1^{\psi} \models \psi$. This situation is captured by a last new transition $covering^{\mathsf{EF}\psi}$, which moves the token from $cover^{\psi}$ to $cover^{\mathsf{EF}\psi}$. Thus, by semantics of EF, $cover^{\mathsf{EF}\psi}$ can be marked iff $N^{\mathsf{EF}\psi}, M_0^{\mathsf{EF}\psi} \models \psi$ (see Fig. 6 in App. B for a full example). From decidability of ρ-PN coverability we obtain:

Theorem 5.3. *For each finite place set P, P-ρCTL-MC is decidable.*

6 DECIDABILITY OF uCDP MODEL CHECKING

In this section we provide an intuitive account on how to reduce uCDP model checking to ρ-PN model checking (the full construction is in App. C). We encode an arbitrary s-state bounded b-uCDP $D = (\Lambda, \mathcal{P}, S_0, m_0)$ with $\Lambda = (\mathcal{S}, \mathcal{I}, \mathcal{T})$, into a ρ-PN $N = (P, T, F)$.

6.1 Configuration Encoding

We use identifiers to represent the domain of DBs: $ID = \Delta \cup \{\bullet\}$. We use places of N (and related markings) to encode configurations of D, reorganized in the following way: *(i)* channel configuration, *(ii)* extension, up to isomorphisms, of the state and input DBs, over a fixed active domain of

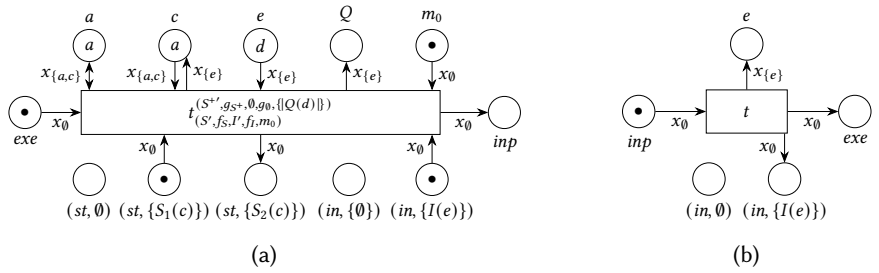


Fig. 4. Places, markings, transitions of the nets (a) in Ex. 6.1 and Ex. 6.3, and (b) in Ex. 6.2.

representative constants, *(iii)* mapping, via a partial function, of the representative constants to the represented state and input DB constants. In fact, let Δ_s and Δ_b be disjoint domains of s and b representative constants, respectively. Each uCDP configuration (S, I, C) is captured by a tuple (S', f_S, I', f_I, C), where S' and I' are state and input DBs with active domain Δ_s and Δ_b, respectively, and the partial functions f_S and f_I map Δ_s and Δ_b to the active domains of S and I, respectively.[10] Moreover, we use the places to represent constants with a special role, i.e., those occurring in S_0 or in $\mathcal{P}$. W.l.o.g., we assume that Δ_s and Δ_b are disjoint from the set Δ_{spec} of such *special constants*. Let $\Delta^* = \Delta_s \cup \Delta_b \cup \Delta_{spec}$.

We now detail how to encode (S', f_S, I', f_I, C) into N. To represent C, we introduce all message symbols in P: a unary message $Q(d)$ in C is encoded by an identifier $d \in ID$ over the *message-place* $Q \in \mathcal{T}$; a propositional message Q in C is encoded by an identifier $\bullet$ over the message-place $Q \in \mathcal{T}$. To represent S', we add to P all state DBs with active domain Δ_s and encode S' by an identifier $\bullet$ over the *DB-place* S'. To represent f_S, we add to P each constant $c \in \Delta_S$ and encode the assignment $f_S(c) = d$ by putting an identifier d on the *constant-place* c. The DB I' and the function f_I are represented analogously. We also encode Δ_{spec} by representing the identity function $id_{\Delta_{spec}}$ of Δ_{spec}: a special constant c is encoded by an identifier c over the *special-constant-place* c. Note that P and ID are not necessarily disjoint.

Summarising, to encode configurations, it is sufficient to use a set P' of places s.t. $P' = \mathcal{T} \cup \{st\} \times H(\Delta_s, \mathcal{S}) \cup \Delta_s \cup \{in\} \times H(\Delta_b, \mathcal{I}) \cup \Delta_b \cup \Delta_{spec}$, where st and in are strings and $H(A, B)$ denotes the set of all DBs with active domain A over the signature B. The place set P is obtained by adding two additional places useful to encode transitions: $P = P' \cup \{exe, inp\}$.

Example 6.1. Let $D = (\Lambda, \mathcal{P}, \{S_1(a)\}, m_0)$ be a 1-state bounded 1-uCDP where $\Lambda = (\{S_1/1, S_2/1\}, \{I/1\}, \{m_0/0, Q/1\})$. Fix $\Delta_s = \{c\}$ and $\Delta_b = \{e\}$. Assume that no constant occurs in $\mathcal{P}$, thus, $\Delta_{spec} = \{a\}$. D has an initial configuration $(\{S_1(a)\}, \{I(d)\}, \{|m_0|\})$, for some $d \in \Delta$, encoded by the marking M_0 in Fig. 4a.

6.2 Step Encoding

We use the transitions of N to encode the steps of D, organized as follows: *(i)* input update, and *(ii)* D2C computation, including message reception, casting, and state DB update.

We update the input DB only when the respective places (those places in $\Delta_b \cup H(\Delta_b, \mathcal{I})$) are empty and inp contains an identifier $\bullet$. In this case, we have to place a single identifier $\bullet$ over a single DB-place in $\{st\} \times H(\Delta_b, \mathcal{I})$, non-deterministically chosen, and mark the constant places in Δ_b with pairwise disjoint (except for $\bullet$) identifiers, again non-deterministically chosen. We denote such choice by a pair (I, f), where $I \in H(\Delta_b, \mathcal{I})$ and f is an injective partial function $\Delta_b \rightharpoonup ID$.

[10] f_S and f_I are partial because S' and I' may have less then s and b constants.

For each $c \in \Delta^*$ and partial functions $f, g : \Delta^* \rightharpoonup \Delta$, let $[c]_g^f = \{c' \in \Delta^* \mid f(c') = g(c)\}$ (if $g(c)$ is not defined, then $[c]_g^f = \emptyset$). Each choice is implemented, up to identifier renaming, by an *input-transition* that *(i)* moves $\bullet$ from *inp* to *exe* and I, using the variable $x_\emptyset$, *(ii)* for each $e \in \Delta_b$ places on e either a non-distinguished random identifier, if $f(e)$ is defined, or $\bullet$, otherwise, using the variable $x_{[e]_f^f}$ (if $f(e)$ is not defined, then $x_{[e]_f^f} = x_\emptyset$, i.e., the variable used to handle $\bullet$). Note that, even if there are infinitely many choices, the transitions capturing them, up to identifier renamings, are finitely many. Thus, we can implement all choices with a finite set T' of *input-transitions*.

Example 6.2. Continuing Ex. 6.1, the choices (I, f_1) and (I, f_2) s.t. $I = \{I(e)\}$, $f_1(e) = a$, and $f_2(e) = d$, for some $d \in \Delta$, are captured, up to identifier renaming, by the transition t in Fig. 4b.

We simulate D2C computations in between of input-transitions, i.e., when *exe* contains a token $\bullet$, each constant-place contains a single identifier, and only one state and one input DB-place contain an identifier $\bullet$. To capture D2C computations, say with input (S, I, m), encoded by (S', f_S, I', F_I, C), and output (S^+, M^+), we build an infinite set T'' of *execution-transitions*. Specifically, first, we introduce a new representative constant q_* (dedicated to m) and extend $f^* = f_S \cup f_I \cup id_{\Delta_{spec}}$ so that $f^*(q_*)$ is either the constant in m, if m is unary, or undefined, otherwise. Second, for each encoding (S', f_S, I', f_I, m) of (S, I, m) and $(S^{+\prime}, g_{S^+}, \emptyset, g_\emptyset, M^+)$ of $(S^+, \emptyset, M^+)$, we add to T' the transition $t_{(S', f_S, I', f_I, m)}^{(S^{+\prime}, g_{S^+}, \emptyset, g_\emptyset, M^+)}$ s.t. *(i)* it is enabled on any marking encoding a tuple (S', h_S, I', h_I, M) where, for each $c \in \Delta^*$, $[c]_{h^*}^{h^*} = [c]_{f^*}^{f^*}$ and $m \in Supp(M)$, and *(ii)* reaches the marking encoding $(S^{+\prime}, g_{S^+}, \emptyset, g_\emptyset, (M - \{\!|m|\!\}) + M^+)$, which, in turn, disregarding the input, encodes the configuration reached by D from $(f_S(S'), f_I(I'), M)$ when delivering m.

Thus, $t_{(S', f_S, I', f_I, m)}^{(S^{+\prime}, g_{S^+}, \emptyset, g_\emptyset, M^+)}$ *(i)* consumes $\bullet$ from I' and moves it from *exe* to *inp*, via the variable $x_\emptyset$, *(ii)* for each constant-place c, consumes either the identifier $f_S(c)$, if defined, or $\bullet$, otherwise, via the variable $x_{[c]_{f^*}^{f^*}}$, *(iii)* for each constant-place c, produces the identifier $g_{S^+}(c)$, if defined, or $\bullet$, otherwise, via the variable $x_{[c]_{g^*}^{f^*}}$ (with $g^* = g_{S^+} \cup g_\emptyset \cup id_{\Delta_{spec}}$), *(iv)* consumes either the identifier d from the message-place Q, if m is the unary message $Q(d)$, or $\bullet$ from Q, if m is the propositional message Q, via the variable $x_{[q_*]_{f^*}^{f^*}}$, and *(v)* for each message $n \in M^+$, produces either the identifier d in the message-place Q, if m is the unary message $Q(d)$, or $\bullet$ in Q, if m is the propositional message Q, via the variable $x_{f^{*-1}(d)}$. Note that there is some $c \in \Delta_s$ s.t. $g^*(c) = d$ and $x_{f^{*-1}(d)} = x_{[c]_{g^*}^{f^*}}$.

Example 6.3. Continuing Ex. 6.1, say that the D2C program computes, on the input $(\{S_1(a)\}, \{I(d)\}, m_0)$, the output $(\{S_2(d)\}, \{\!|Q(d)|\!\})$. This computation is captured by the transition $t_{(S', f_S, I', f_I, m_0)}^{(S^{+\prime}, g_{S^+}, \emptyset, g_\emptyset, \{\!|Q(d)|\!\})}$, depicted in Fig. 4a, s.t. $S' = \{S_1(c)\}$, $S^{+\prime} = \{S_2(c)\}$, $I' = \{e\}$, $f^*(a) = g^*(a) = a$, $f^*(c) = f^*(e) = g^*(c) = d$, and $f^*(q_*)$, $g^*(q_*)$, and $g^*(e)$ are undefined.

Each uCDP transition is simulated by two successive input- and execution transitions and vice-versa. However, we cannot add T'' to T, because T must be finite, while T'' is infinite. Nevertheless, all the transitions in T'' have conditions taken from a finite set of possibilities, accounting for S', I', S^+, Q, and the classes $[c]_{f^*}^{f^*}$ and $[c]_{g^*}^{f^*}$, for each $c \in \Delta^*$. Since Δ^* is finite, there are finitely many such possibilities. Specifically, fixed S', I', and message m over the relation Q, all transitions $t_{(S', f_S, I', f_I, m)}^{(S^{+\prime}, g_{S^+}, \emptyset, g_\emptyset, M^+)}$ with the same classes $[c]_{f^*}^{f^*}$ and $[c]_{g^*}^{f^*}$, for each $c \in \Delta^*$, have the same conditions. Thus, they can be abstracted away by a single transition $t(S', I', S^{+\prime}, Q, \sim, h)$, where $\sim$ is a binary symmetric and transitive relation on Δ^*, capturing the classes $[c]_{f^*}^{f^*}$, and h is a partial function $\Delta_s \rightharpoonup \{[c]_\sim \mid c \in \Delta^*\}$. Let T''' be the finite set of transitions abstracting T''. The set T of transitions of N is the finite set $T' \cup T'''$ and the flow-function F is defined as described above.

Theorem 6.4. *D and $N = (P, T, F)$ are bisimilar under a bisimulation B that expands each step of D into two steps of N.*

6.3 Formula Encoding

We encode a formula $\varphi \in \mathsf{EF}(\mathsf{EF}^+, bool, st)$ into P-ρCTL. By Theorem 6.4, it is enough to handle FO formulas. W.l.o.g., we assume that all constants occurring in φ are special constants in Δ_{spec}. Also, since φ ranges only over the state signature, it is satisfied by any configuration (S, I, C) s.t. $\varphi \models S$. Since D is s-state bounded, up to isomorphism there is only a finite set $\{S_1, \ldots, S_k\}$ of state DBs satisfying φ. Thus, φ is equivalent to the disjunction of markings m_i encoding the configurations $(S_i, \emptyset, \emptyset)$, for each $i \in \{1, \ldots, k\}$.[11]

Similarly, we can encode also formulas ψ from $\mathsf{EF}(\mathsf{EF}^+, pos, st{+}ch)$. In fact, since ψ is positive, it can only assert the existence of a fixed number of tuples in the state or in the channel. Thus, because of state boundedness, up to isomorphisms and channel inclusion there is only a finite set $\{(S_1, \emptyset, C_1), \ldots, (S_k, \emptyset, C_k)\}$ of configurations satisfying ψ. Thus, ψ is equivalent to the disjunction of markings m_i encoding the configurations $(S_i, \emptyset, C_i)$, for each $i \in \{1, \ldots, k\}$.

Theorem 6.5. *$\mathsf{EF}[\mathsf{EF}^+, bool, st]$-MC and $\mathsf{EF}[\mathsf{EF}^+, pos, st{+}ch]$-MC are decidable.*

7 UNDECIDABILITY OF uCDP MODEL CHECKING

We show that extending uCDP with some natural feature makes verification of $\mathsf{EF}[-, pos, st]$ undecidable. Then, we show how some extension of $\mathsf{EF}[-, pos, st]$ allows to lift them.

7.1 Extended uCDP

We consider two features missing in the standard uCDP model, i.e., *termination-awareness* and the availability of *historically fresh input*. Termination-awareness is the ability for the uCDP to signal through a dedicated flag in the state DB that the node cannot be activated anymore and no error has been deduced. Due to the activation policy, this can happen only when all channels are empty. Since nodes have no explicit mechanism to check termination, this feature requires a modification of the semantics. Formally, a *termination-aware uCDP* is a uCDP with the provision that, on top of the standard semantics, for each configuration (S, I, C), (1) S can host a special `terminated` flag that cannot be used in D2C rules, (2) the flag is in S iff S does not contain the `error` flag and C is empty.

The second feature is the availability of historically fresh constants. A constant c is *historically fresh w.r.t. a finite run* if, for each configuration σ in the run, the constant never appears in σ. Since fresh constants can only come from the input DB, the availability of historically fresh constants is a feature of the input-policy. Even in this case we have to modify the semantics. First, we label the initial configuration with its active domain. Then we proceed in steps as with the standard semantics, but, at each iteration, we require that (1) the active domain of the new input DB has at least one constant not in the set labeling the previous configuration and (2) the new configuration is labeled by the union of its own active domain and of the label of the previous configuration. A *fully-extended* (FE) uCDP is a termination-aware uCDPs with historically fresh constants availability.

7.2 EF Undecidability for Fully-extended uCDPs

2-counters machines (2CM) [22] are finite state automata operating over two counters. Technically, a 2CM is a tuple $(Q, inst, q_0, q_f)$ where Q is a set of states, $q_0, q_f \in Q$ are the initial and finite state, respectively, and *inst* is the set of instructions. These can be of two types: increment or conditional

[11] Note that, if M_1 covers $(S, \emptyset, \emptyset)$ and M_1 encodes the configuration (S_1, I_1, C_1) , then S_1 is isomorphic to S and $I_1 = \emptyset$, while C_1 may be anything.

decrement instructions. An increment instruction is a tuple $(q, +, i, q')$. It can fire only when the current state is q, resulting in the state q' while increasing counter i by one. A decrement instruction is a tuple $(q, -, i, q', q'')$. It can fire only when the current state is q and has one of the following two outcomes. If counter i is non-zero, then the new state is q' and counter i is decreased by one. If counter i is zero, then the new state is q'' and counter i is not updated. It is well known that termination of 2CM (reachability of q_f) is undecidable even for deterministic 2CMs, i.e., when for each $q \in Q$ there is at most one instruction fireable on q.

Given a 2CM $\mathcal{M} = (Q, inst, q_0, q_f)$, we build an FE uCDP C and an $\mathsf{EF}[-, pos, st]$ formula that checks whether the 2CM terminates. We use the uCDP signature $\Lambda = (\mathcal{S}, \mathcal{I}, \mathcal{T})$ where $\mathcal{S} = \{\mathtt{q}/0 \mid \mathtt{q} \in Q\} \cup \{\mathtt{error}/0, \mathtt{someInput}/0, \mathtt{curr}_1/1, \mathtt{curr}_2/1, \mathtt{zGuess}/0, \mathtt{check}/0\}$, $\mathcal{I} = \{\delta/1\}$, and $\mathcal{T} = \{\mathtt{cnt}_1/1, \mathtt{cnt}'_1/1, \mathtt{cnt}_2/1, \mathtt{cnt}'_2/1, \mathtt{empty}/0, \mathtt{start}/0\}$.

The current 2CM machine state q is encoded in the current state DB by means of a flag $\mathtt{q}/0$. Thus, the initial state DB is $\{q_0\}$. That way, if a D2C rule π is involved in the simulation of the unique transition fireable over the state q, it is convenient to put a `prev q` in the body of π, so that π will not fire upfront any other $\mathtt{q}'$.

Given a family F of input DBs (messages) and a machine state q, we say that the node *expects* an input DB (message) in F when in state q if the reception of an input DB (message) not in F when the state flag is q triggers an `error`. In this uCDP C, for each state q, the node expects that the input contains only one constant, stored in the input predicate $\delta/1$. This can be enforced by the rules:

```
someInput if δ(x).
error if not someInput.
error if δ(x), δ(y), x≠y.
```

Since the uCDP is FE, the node implicitly always expects to receive from the input a single, historically fresh constant.

For $i = 1, 2$, the counter i of the 2CM is encoded by the sum of the multiplicities of two messages $\mathtt{cnt}_i/1$ and $\mathtt{cnt}'_i/1$ tagged with the constant t that is mentioned in the state predicate $\mathtt{curr}_i$. We need two *cnt* predicates because nodes send messages in sets, instead of multisets. By doing so, we can still receive one message (which would decrease the counter) and send back two messages with the same constant (globally increasing the counter by one). Thus, by changing the constant in $\mathtt{curr}_i$, it is possible to reset the representation of counter i to zero, while retaining all remaining $\mathtt{cnt}_i/1$ and $\mathtt{cnt}'_i/1$ messages on the channel.

At startup the node stores in predicates $\mathtt{curr}_i/1$ the constant from the input and sends on the channel a message $\mathtt{empty}/0$. This message must always be on the channel each time the state machine is not q_f, to make sure that the node can react and simulate a counter machine transition even when the channel is empty and, thus, the counters are both set to zero.

```
empty if start.
curr_i(X) if start, δ(X).
```

Since the channel is unordered, this message could be received even when there are still other messages on the channels. This issue will be handled by performing zero guesses (maybe wrongly).

To perform an increment $(q, +, i, q')$, the node sends back the received message along with a new one containing the stored constant. For $j \neq i$:

```
cnt_i(X) if empty, curr_i(X) prev q.
empty if empty prev q.
cnt_i(X) if cnt_j(Y), curr_i(X) prev q.
cnt_j(Y) if cnt_j(Y) prev q.
cnt_i(X) if cnt_j'(Y), curr_i(X) prev q.
cnt_j'(Y) if cnt_j'(Y) prev q.
cnt_i(X) if cnt_i'(X) prev q.
cnt_i'(X) if cnt_i'(X) prev q.
cnt_i'(X) if cnt_i(X) prev q.
cnt_i(X) if cnt_i(X) prev q.
q' if prev q.
```

To perform a conditional decrement $(q, -, i, q', q'')$, the node considers the incoming message. If it represents a token in the counter i with the constant mentioned in $\texttt{curr}_\texttt{i}$, then it performs the decrement, by consuming the incoming message and changing the state:

```
q' if cnt_i(X), curr_i(X) prev q.
q' if cnt'_i(X), curr_i(X) prev q.
```

If the message represents a token in the counter i, but the constant is not mentioned in $\texttt{curr}_\texttt{i}$, then an error is deduced:[12]

```
error if cnt_i(X), curr_i(Y) prev q, X≠Y.
error if cnt_i'(X), curr_i(Y) prev q, X≠Y.
```

Otherwise, it guesses (maybe wrongly) that the counter is empty and changes the state accordingly. The flag zGuess/0 signals that the node has performed a zero guess. For $j \neq i$:

```
empty if empty prev q.
zGuess if empty prev q.
cnt_j(X) if cnt_j(X) prev q.
```

```
zGuess if cnt_j(X) prev q.
cnt'_j if cnt'_j(X) prev q.
zGuess if cnt'_j(X) prev q.
q'' if zGuess prev q.
```

The zero guess triggers the update of the constant stored in $\texttt{curr}_\texttt{i}$.

```
curr_i(X) if zGuess, δ(X) prev q.
```

Notice that this is the only case in which the constant in $\texttt{curr}_\texttt{i}$ must be updated. Thus, we add a last rule that makes $\texttt{curr}_\texttt{i}$ persistent unless a zero guess is made.

```
curr_i(X) if not zGuess, curr_i(X) prev q.
```

With these rules, the node can deduce the $\texttt{q}_\texttt{f}$ flag either performing only correct zero-guesses or some wrong zero-guess. In the former case, the node has faithfully simulated the 2CM, thus witnessing its termination. In the latter case, the node has cheated and its result must be ignored. This service is enforced by an additional phase of the computation, which checks, with the help of termination-awareness, whether a wrong guess had been made. This phase has to check whether there is a $\texttt{cnt}_\texttt{i}$ ($\texttt{cnt}_\texttt{i'}$) message in the channel with a constant not in $\texttt{curr}_\texttt{i}$. The reach of $\texttt{q}_\texttt{f}$ triggers the consumption of the channels along with a comparison of the incoming message with $\texttt{curr}_\texttt{i}$, for the relevant i. Specifically, the reach of $\texttt{q}_\texttt{f}$ triggers the deduction of the check/0 state, which is persistent.

```
check if prev q_f.
check if prev check.
```

Then, the messages are consumed, one per step, and compared as needed. If a comparison detects a deprecated constant in the messages, an error is deduced. Otherwise it continues until the flag terminated is risen.

```
error if cnt_1(X), curr_1(Y), X≠ Y if check.
error if cnt'_1(X), curr_1(Y), X≠ Y if check.
```

```
error if cnt'_2(X), curr_2(Y), X≠ Y if check.
error if cnt'_2(X), curr_2(Y), X≠ Y if check.
```

By adding these rules, after reaching $\texttt{q}_\texttt{f}$, any possible wrong zero-guess is detected. Moreover, there is a reception order that results in an error iff all reception orders result in an error. Thus, terminated is eventually risen iff no wrong guess was made. Hence, the formula EF(check $\wedge$ terminated) checks whether there is at least one run that witnesses the termination of the 2CM without making wrong guesses. This means that this formula is true iff the 2CM terminates.

Theorem 7.1. EF$[-, pos, st]$-MC *for FE uCDPs is undecidable.*

[12] As it will be explained later, this means that some wrong zero guess has been done.

7.3 Lifting Historical Freshness

Historical freshness can be checked by storing all the constants read from the input, with multiplicity, and then checking whether there is at least one repeated constant. Since the channel is the only component that can store an unbounded multi-set, we have to make sure that, each time a constant is taken from the input, it is sent on the channel in an additional `received/1` message. The node triggers an error if these messages are received before the check phase.

```
36 received(X) if curr1(X), zGuess prev not check, not qf.
37 received(X) if curr2(X), zGuess prev not check, not qf.
38 error if received(X) prev not check.
```

After reaching q_f (recall rules 30 and 31), the check is performed along parallel runs: the constant in the first received message `received` is stored in the state and compared to all constants arriving via `received` messages.

```
39 freshCheck(X) if received(X) prev check, not freshChecking.
40 freshChecking if freshCheck(X).
41 freshCheck(X) if prev freshCheck(X).
42 error if received(X) prev freshCheck(X).
```

We now have to adapt the verification formula to check all parallel runs at once. A wrong fresh-guess was made iff there is at least one run that delivers to the node two identical instances of `received` right after the `check` phase begun. This can be done by any CTL operator employing an A:[13]

$$\mathsf{EF}(\mathsf{q_f} \wedge (\mathsf{AG}\, \neg \mathtt{error})) \tag{1}$$

$$\mathsf{EF}(\mathsf{q_f} \wedge (\mathsf{AXAX}\, \neg \mathtt{error})) \tag{2}$$

$$\mathsf{EF}(\mathsf{q_f} \wedge (\mathsf{AF}\, \mathtt{terminated})) \tag{3}$$

$$\mathsf{EF}(\mathsf{q_f} \wedge (\mathsf{A}(\top\, \mathsf{U}\, \mathtt{terminated})) \tag{4}$$

7.4 EF Undecidability for Standard uCDPs

Remarkably, Formulas (1) and (2) make no use of the flag `terminated`. This is because, once the check phase starts, as long as no error can be deduced globally on all runs or, equivalently, no error can be deduced after two steps, since the channel is consumed step after step, each run will reach a configuration where the channel is empty and no error is deduced. Thus, these two formulas hold even on standard uCDP (without termination awareness) iff the 2CM terminates. Moreover, by adding the rule

```
43 correct if not error.
```

we can substitue the $\neg$`error` literal in the two formulas above with the atom `correct`, and check 2CM termination via a formula in one of the fragments $\mathsf{EF}[\mathsf{AG}, pos, st]$ or $\mathsf{EF}[\mathsf{AX}^2, pos, st]$.

Theorem 7.2. $\mathsf{EF}[\mathsf{AG}, pos, st]$-MC *and* $\mathsf{EF}[\mathsf{AX}^2, pos, st]$-MC *over uCDPs are undecidable.*

We now adapt the previous solution so as to check 2CM termination by using an $\mathsf{EF}[-, bool, st{+}ch]$ formula. Given a configuration C, we can specify a formula ψ that checks whether the messages $\mathtt{cnt}_i$ and $\mathtt{cnt}'_i$ on the channel comply to the content of $\mathtt{curr_i}$, as follows: $\forall \mathsf{x}\, ((\mathsf{c_1}(\mathsf{x}) \vee \mathsf{c'_1}(\mathsf{x}) \rightarrow \mathtt{curr_1}(\mathsf{x})) \wedge (\mathsf{c_2}(\mathsf{x}) \vee \mathsf{c'_2}(\mathsf{x}) \rightarrow \mathtt{curr_2}(\mathsf{x})))$. We can thus lift both termination-awareness and historical freshness by checking whether there is a run such that ψ holds on the first configuration

[13]Notice that we are lifting only historical freshness and still resort to termination-awareness.

that reaches q_f. To do this, we have to add a rule to signal a propositional flag `justReached` only the first time q_f is reached (the rule is correct, since previous rules make q_f persistent):

```
44 justReached if q_f prev not q_f .
```

Then, we have to check the formula only when reaching `justReached`, i.e., we have to check $\mathsf{EF}(\texttt{justReached} \wedge \psi)$. This formula can be rewritten in $\mathsf{EF}[-, \mathit{bool}, \mathit{st{+}ch}]$ via standard Boolean equivalences and quantification dualities.

Theorem 7.3. $\mathsf{EF}[-, \mathit{bool}, \mathit{st{+}ch}]$-MC *over uCDPs is undecidable.*

8 CONCLUSIONS

We recap (un)decidability for $\mathsf{EF}[\Box, \triangle, \circ]$-MC on uCDPs:

$\Box$	–	–	EF^+	EF^+	AG	AX^2
$\circ$ $\triangle$	*pos*	*bool*	*pos*	*bool*	*pos*	*pos*
st	D	D	D	D	U	U
st+ch	D	U	D	U	U	U

The properties refer to branching-time first-order properties verified over CDPs where nodes asynchronously exchange messages with at most a unary signature, the state of each node has a bounded size, while communication channels have unbounded capacity. Our results are obtained by encoding such verification problems into corresponding coverability verification problems over ρ-PNs. In spite of the extremely high complexity in the analysis of such nets, several effective techniques and tools for the (symbolic) exploration of their state space exists (see, e.g., [17, 27]). Our work consequently paves the way towards the application of such techniques to the practical analysis of declarative distributed systems. Another natural follow-up is to continue the fine-grained analysis here provided by considering different forms of communication, as well as linear-time properties, for which decidability results over state-bounded data-aware processes exist [11], but have never been studied when part of the overall system configuration is unbounded.

ACKNOWLEDGMENTS

This research has been partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation, by the Province of Bolzano and DFG through the project D2G2 (DFG grant n. 500249124), by the HEU project CyclOps (under GA n. 101135513), by the UNIBZ project ADAPTERS, and by the PRIN MUR project PINPOINT (Prot. 2020FNEB27).

REFERENCES

[1] Serge Abiteboul, Émilien Antoine, Gerome Miklau, Julia Stoyanovich, and Jules Testard. 2013. Rule-based Application Development Using Webdamlog. In *Proc. of the 34th ACM Int. Conf. on Management of Data (SIGMOD)*. ACM, 965–968. https://doi.org/10.1145/2463676.2465251

[2] Peter Alvaro, Tom J. Ameloot, Joseph M. Hellerstein, William Marczak, and Jan Van den Bussche. 2011. *A Declarative Semantics for Dedalus*. Technical Report UCB/EECS-2011-120. EECS Department, University of California, Berkeley. http://www.eecs.berkeley.edu/Pubs/TechRpts/2011/EECS-2011-120.html

[3] Tom J. Ameloot and Jan Van den Bussche. 2014. Positive Dedalus Programs Tolerate Non-causality. *J. of Computer and System Sciences* 80, 7 (2014), 1191–1213. https://doi.org/10.1016/j.jcss.2014.01.005

[4] Babak Bagheri Hariri, Diego Calvanese, Giuseppe De Giacomo, Alin Deutsch, and Marco Montali. 2013. Verification of Relational Data-Centric Dynamic Systems with External Services. In *Proc. of the 32nd ACM Symp. on Principles of Database Systems (PODS)*. 163–174. https://doi.org/10.1145/2463664.2465221

[5] Babak Bagheri Hariri, Diego Calvanese, Marco Montali, and Alin Deutsch. 2014. State-Boundedness in Data-Aware Dynamic Systems. In *Proc. of the 14th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR)*. AAAI Press, 458–467.

[6] Christel Baier and Joost-Pieter Katoen. 2008. *Principles of Model Checking*. The MIT Press.

[7] Francesco Belardinelli, Alessio Lomuscio, and Fabio Patrizi. 2011. Verification of Deployed Artifact Systems via Data Abstraction. In *Proc. of the 9th Int. Joint Conf. on Service Oriented Computing (ICSOC) (Lecture Notes in Computer Science, Vol. 7084)*. Springer, 142–156.

[8] Hans-Dieter Burkhard. 1982. On Fairness in Petri Nets. *Rostocker Mathematisches Kolloquium* (1982), 85–96. https://doi.org/10.18452/9417

[9] Diego Calvanese, Giuseppe De Giacomo, and Marco Montali. 2013. Foundations of data-aware process analysis: a database theory perspective. In *Proc. of the 32nd ACM Symp. on Principles of Database Systems (PODS)*. ACM, 1–12. https://doi.org/10.1145/2463664.2467796

[10] Diego Calvanese, Giuseppe De Giacomo, Marco Montali, and Fabio Patrizi. 2018. First-order mu-calculus over Generic Transition Systems and Applications to the Situation Calculus. *Information and Computation* 259, 3 (2018), 328–347. https://doi.org/10.1016/j.ic.2017.08.007

[11] Diego Calvanese, Giuseppe De Giacomo, Marco Montali, and Fabio Patrizi. 2022. Verification and Monitoring for First-Order LTL with Persistence-Preserving Quantification over Finite and Infinite Traces. In *Proc. of the 31st Int. Joint Conference on Artificial Intelligence*. IJCAI Org., 2553–2560.

[12] Diego Calvanese, Francesco Di Cosmo, Jorge Lobo, and Marco Montali. 2021. Convergence Verification of Declarative Distributed Systems. In *Proc. of the 36th Italian Conf. on Computational Logic (CILC 2021) (CEUR Workshop Proceedings, Vol. 3002)*. CEUR-WS.org, 62–76.

[13] Giuseppe De Giacomo, Yves Lespérance, and Fabio Patrizi. 2016. Bounded Situation Calculus Action Theories. *Artificial Intelligence* 237 (2016), 172–203. https://doi.org/10.1016/j.artint.2016.04.006

[14] Alin Deutsch, Richard Hull, Yuliang Li, and Victor Vianu. 2018. Automatic Verification of Database-centric Systems. *ACM SIGLOG News* 5, 2 (2018), 37–56.

[15] Alin Deutsch, Liying Sui, and Victor Vianu. 2007. Specification and Verification of Data-Driven Web Applications. *J. of Computer and System Sciences* 73, 3 (2007), 442–474.

[16] Alin Deutsch, Liying Sui, Victor Vianu, and Dayou Zhou. 2006. Verification of Communicating Data-Driven Web Services. In *Proc. of the 25th ACM Symp. on Principles of Database Systems (PODS)*. 90–99. https://doi.org/10.1145/1142351.1142364

[17] Silvio Ghilardi, Alessandro Gianola, Marco Montali, and Andrey Rivkin. 2022. Petri Net-based Object-centric Processes with Read-only Data. *Information Systems* 107 (2022), 102011.

[18] Joseph M. Hellerstein. 2010. The Declarative Imperative: Experiences and Conjectures in Distributed Logic. *SIGMOD Record* 39, 1 (2010), 5–19.

[19] Richard M. Karp and Raymond E. Miller. 1969. Parallel Program Schemata. *J. of Computer and System Sciences* 3, 2 (1969), 147–195.

[20] Ranko Lazic, Thomas Christopher Newcomb, Joël Ouaknine, A. W. Roscoe, and James Worrell. 2008. Nets with Tokens which Carry Data. *Fundamenta Informaticae* 88, 3 (2008), 251–274.

[21] Jiefei Ma, Franck Le, Alessandra Russo, and Jorge Lobo. 2016. Declarative Framework for Specification, Simulation and Analysis of Distributed Applications. *IEEE Trans. on Knowledge and Data Engineering* 28, 6 (2016), 1489–1502.

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

[23] Fernando Rosa-Velardo. 2017. Ordinal Recursive Complexity of Unordered Data Nets. *Information and Computation* 254 (2017), 41–58.

[24] Fernando Rosa-Velardo and David de Frutos-Escrig. 2011. Decidability and Complexity of Petri Nets with Unordered Data. *Theoretical Computer Science* 412, 34 (2011), 4439–4451. https://doi.org/10.1016/j.tcs.2011.05.007

[25] Dmitry Solomakhin, Marco Montali, Sergio Tessaris, and Riccardo De Masellis. 2013. Verification of Artifact-Centric Systems: Decidability and Modeling Issues. In *Proc. of the 11th Int. Joint Conf. on Service Oriented Computing (ICSOC) (Lecture Notes in Computer Science, Vol. 8274)*. Springer, 252–266.

[26] Victor Vianu. 2009. Automatic Verification of Database-Driven Systems: a New Frontier. In *Proc. of the 12th Int. Conf. on Database Theory (ICDT)*. 1–13. https://doi.org/10.1145/1514894.1514896

[27] Michael Westergaard, Sami Evangelista, and Lars Michael Kristensen. 2009. ASAP: An Extensible Platform for State Space Analysis. In *Proc. of the 30th Int. Conf. on Application and Theory of Petri Nets (PETRI NETS) (Lecture Notes in Computer Science, Vol. 5606)*. Springer, 303–312.

A UCDP EXAMPLE

The CDP in Ex. 3.10 uses unary messages and is channel bounded but not state bounded. Thus, technically, it is not a uCDP (see Sec. 4). We now provide a variant that is also state bounded and, thus, an uCDP.

Example A.1. Fig. 5 contains the program $\mathcal{P}$ of a 1-state, 1-input bounded uCDP $D = (\Lambda, \mathcal{P}, \emptyset, \mathsf{none})$ that models a single landing pad in a drone base. The signature is $\Lambda = (\mathcal{S}, \mathcal{I}, \mathcal{T})$ where $\mathcal{S} = \{\mathsf{pad}/1, \mathsf{incoming}/0, \mathsf{free}/0, \mathsf{occupied}/0\}$, $\mathcal{I} = \{\mathsf{addDrone}/1\}$, $\mathcal{T} = \{\mathsf{none}/0, \mathsf{drone}/1\}$. At each step, if present, the drone occupying the pad during the previous step has to take off (rule 4). A flying drone of a given type x is encoded by a message $\mathsf{drone}(\mathsf{x})$. several drones may have the same type, but the overall number of types and of drones remains unbounded. A landing drone is simulated by the reception of a $\mathsf{drone}(\mathsf{x})$ message, which results in the drone occupying the pad (rule 1) and an $\mathsf{incoming}$ flag to be risen (rule 2). The absence of landing drones is simulated by the reception of the none message, which results in its immediate recasting (rule 5). New drones can be registered by external services one by one (according to 1-input boundedness). The attempted registration of a new drone of type x is encoded by the input DB $\{\mathsf{addDrone}(\mathsf{x})\}$. If there is no landing drone, then, the drone is placed on the pad (rule 3); otherwise, assuming that landing drone takes precedence on new drones, the registration fails and has no effect. At the end of each step, the node also explicitly signals whether the pad is occupied or empty (rules 6 and 7). Since drones can be registered only as long as all other drones are on duty, the uCDP is channel unbounded (even if state and channel bounded).

In the following, we list some verification sentence for the uCDP in Ex.A.1. The $\mathsf{EF}[-, pos, st]$ sentence

$$\mathsf{EF}\,\exists \mathsf{X}.\,\mathsf{pad}(\mathsf{X})$$

asks whether it is possible to reach a configuration where there is some drone on the pad. The sentence is clearly satisfied by the uCDP. The similar $\mathsf{EF}[-, pos, st{+}ch]$ sentence

$$\mathsf{EF}\,\exists \mathsf{X}.\,(\mathsf{pad}(\mathsf{X}) \wedge \mathsf{drone}(\mathsf{X}))$$

asks whether it is possible to have a landed drone and a flying drone of the same type. The sentence is satisfied. Adding negation, the $\mathsf{EF}[-, bool, st{+}ch]$ sentence

$$\mathsf{EF}\,\neg\exists \mathsf{X}.\,(\mathsf{pad}(\mathsf{X}) \wedge \mathsf{drone}(\mathsf{X}))$$

asks whether we can reach a configuration were the landed drone (if any) and the flying drones have different types. The sentence is satisfied. Adding further EF operators, the $\mathsf{EF}[\mathsf{EF}^{+}, pos, st]$ sentence

$$\mathsf{EF}\,(\mathsf{free} \wedge (\mathsf{EF}\,\mathsf{occupied}))$$

asks whether it is possible to reach a configuration where the landing pad is free and after some additional computation steps it is occupied. The sentence is satisfied. By adding negation and a

```
1 pad(X) if drone(X).
2 incoming if drone(X).
3 pad(X) if addDrone(X), not incoming.
4 drone(X) if prev pad(X).
5 none if none.
6 occupied if pad(X).
7 free if pad(X).
```

Fig. 5. Program of the uCDP in Ex. A.1.

query on the channels, the $\mathsf{EF}[\mathsf{EF}^+, \mathit{bool}, \mathit{st}{+}\mathit{ch}]$ sentence

$$\mathsf{EF}\,(\mathtt{free} \wedge (\mathsf{EF}\,\mathtt{occupied} \wedge (\mathsf{EF}\,\mathtt{free} \wedge \neg\exists \mathtt{X}.\,\mathtt{drone}(\mathtt{X}))))$$

asks whether there is an extension of a computation as above in which no drone is flying. Even this sentence is satisfied. We now focus on the AG operator. Let $\top$ be any tautology, e.g., $\mathtt{free} \vee \neg\mathtt{free}$. The $\mathsf{EF}[(\mathsf{AG}), \mathit{pos}, \mathit{st}{+}\mathit{ch}]$ sentence

$$\mathsf{EF}(\top \wedge \mathsf{AG}\,(\mathtt{free} \vee (\exists \mathtt{X}.\,\mathtt{pad}(\mathtt{X}) \wedge \mathtt{drone}(\mathtt{X}))))$$

asks whether it is possible to reach a configuration from which, forever, either the pad is free or the drone at the pad is of the same type of a currently flying drone. This is never the case, even after several computation steps, since there is no constraint on the number of drones per type. In case free and occupied are not part of the state signature (e.g., if rules 6 and 7 are dropped), we can still rephrase the sentence using negation. The $\mathsf{EF}[(\mathsf{AG}), \mathit{bool}, \mathit{st}{+}\mathit{ch}]$ sentence

$$\mathsf{EF}(\top \wedge \mathsf{AG}\,((\neg\exists \mathtt{X}.\,\mathtt{pad}(\mathtt{X})) \vee (\exists \mathtt{X}.\,\mathtt{pad}(\mathtt{X}) \wedge \mathtt{drone}(\mathtt{X}))))$$

is equivalent to the previous one. By using AX, the $\mathsf{EF}[\mathsf{AX}^2, \mathit{bool}, \mathit{st}{+}\mathit{ch}]$ sentence

$$\mathsf{EF}\,(\mathtt{occupied} \wedge \mathsf{AXAX}\,\neg\exists \mathtt{X}.\,\mathtt{drone}(\mathtt{X}))$$

asks whether it is possible to reach a configuration where there is a landed drone and, after two steps, there is no way of having a flying drone. This sentence is not satisfied. Similar sentences apply to the CDP in Ex. 3.10.

B *P*-ρCTL SEMANTICS

The language *P*-ρCTL is defined in Sec. 5 as follows.

Definition B.1. *Given a set P of places, P-ρCTL is the language of formulas φ defined by the following grammar:*

$$\varphi ::= M \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \mathsf{EF}\varphi$$

where M denotes a marking over the place set P. Markings M are called atomic *P-ρCTL formulas.*

A *P*-ρCTL formula can be entailed or not by a pair of ρ-PN and initial marking. The semantics of *P*-ρCTL is that of CTL, with the provision that atomic formulas M express that the current marking M' covers the atomic formula up to isomorphisms, i.e., there is a permutation h of ID such that, for each $p \in P$, it is true that $h(M)(p) \subseteq M'(p)$, also denoted by $M \preceq M'$.

Definition B.2. *Given a formula φ of P-ρCTL, a ρ-PN $N = (P, T, F)$, and a marking M_0 for N, the net N and marking M_0 entail the formula φ, denoted by $N, M_0 \models \varphi$ if:*

- *if φ is an atomic formula M, then $M_0 \preceq M$;*
- *if $\varphi = \varphi_1 \wedge \varphi_2$, then $N, M_0 \models \varphi_1$ and $N, M_0 \models \varphi_2$;*
- *if $\varphi = \varphi_1 \vee \varphi_2$, then $N, M_0 \models \varphi_1$ or $N, M_0 \models \varphi_2$;*
- *if $\varphi = \mathsf{EF}\varphi_1$, then, there is a sequence $M_0 \to \cdots \to M_n$ of markings, for some $n \in \mathbb{N}$, such that $N, M_n \models \varphi$.*

P-ρCTL model checking can be reduced to coverability.

Example B.3. Consider the ρ-PN N in Figure 3a and the markings M_1 and M_2 such that $M_1(p) = \emptyset$, $M_1(q) = \{\!|d, d|\!\}$, $M_2(p) = \{\!|e|\!\}$, and $M_2(q) = \{\!|e|\!\}$, for some $d, e \in ID$. Let φ be the *P*-ρCTL formula $\mathsf{EF}(M_1 \wedge M_2) \vee M_1$ Let ψ be the formula $\mathsf{EF}(m_1 \wedge m_2) \vee m_3$. The net N^ψ depicted in Fig. 6 can cover from the initial marking M_0^ψ the target marking M_f that puts exactly one black token on cover^ψ (i.e., can reach a marking that extends M_f) if and only if $N, M_0 \models \varphi$.

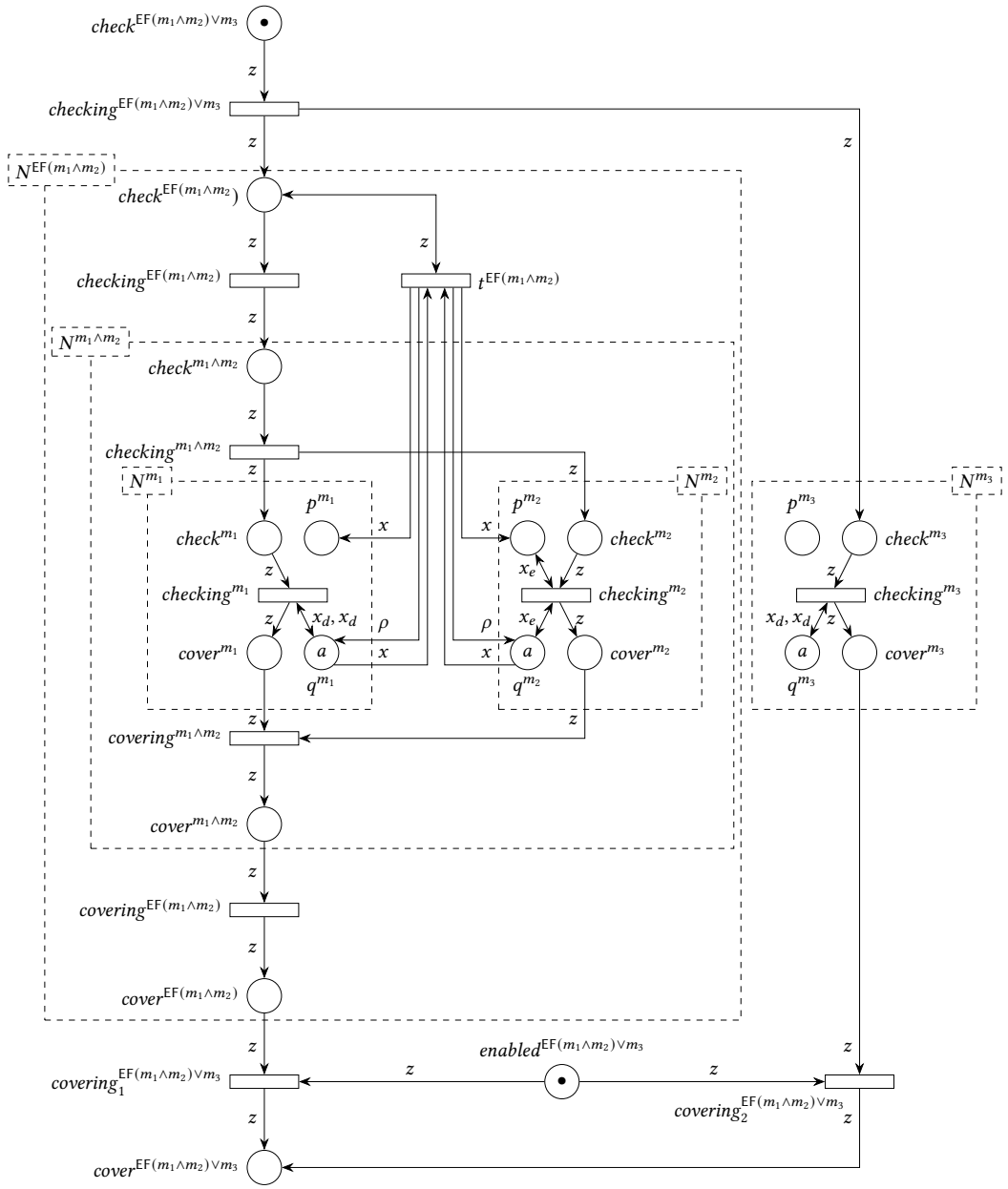


Fig. 6. The ρ-PN N^{ψ} and initial marking M_0^{ψ} in Ex. B.3.

C TECHNICAL RESULTS ON DECIDABILITY

We detail the technical development of the encoding explained in Sec. 6. This provides a reduction from verification of $\mathsf{EF}[\mathsf{EF}^+, bool, st]$-MC and $\mathsf{EF}[\mathsf{EF}^+, pos, st{+}ch]$-MC to verification of P-ρCTL model checking.

C.1 Net Definition

Given an s-state bounded b-uCDP $D = (\Lambda, \mathcal{P}, S_0, m_0)$, where $\Lambda = (\mathcal{S}, \mathcal{I}, \mathcal{T})$, and a verification formula φ, fix the following disjoint sets of constants: $\Delta_s = \{c_1, \ldots, c_s\}$, $\Delta_b = \{e_1, \ldots, e_b\}$ (e stands for *external*, referring to the external inputs), $\{q\}$, and Δ_{spec} of constants occurring in S_0, $\mathcal{P}$, and φ. Let $\Delta^* = \Delta_s \cup \Delta_b \cup \{q_*\}\Delta_{spec}$. Equip Δ^* with a well order $\leq$ such that, for each $a \in \Delta_{spec}$, $i \in \{1, \ldots, s\}$, and $j \in \{1, \ldots, b\}$, $a < c_i < e_j < q_*$.[14]

Given a totally ordered set X, we denote by $\langle X \rangle_i$ the i-th element of X.

Places. The set P of places of N is $P = \mathcal{T} \cup \{st\} \times H(\Delta_s, \mathcal{S}) \cup \Delta_s \cup \{in\} \times H(\Delta_b, \mathcal{I}) \cup \Delta_b \cup \Delta_{spec} \cup \{inp, exe\}$.

Transitions. Formally, the set of transitions contains tuples. To stress that we use these tuples as transition names, we use the notation $t_{(t_1,\ldots,t_n)}$ to denote the tuple $(t_1, \ldots, t_n)$. We denote the set of symmetric and transitive binary relations on a set X by $Rel(X)$. The set T' of input-transitions is $T' = \{t_{(I',\sim)} \mid I' \in H(\Delta_e, \mathcal{I}), \sim \in Rel(\Delta_e)\}$. The set of execution-transitions is abstracted away by the set T''' of transitions $t_{(S',I',S^{+\prime},\sim,h,Q)}$ where

(1) $S', S^{+\prime} \in H(\Delta_s, \mathcal{S})$, $I' \in H(\Delta_e, \mathcal{I})$, $Q \in \mathcal{T}$,
(2) $\sim \in Rel(\Delta^*)$ and $q_* \sim q_*$ iff Q is unary,
(3) $h : \Delta_s \rightharpoonup \{[c]_\sim \mid c \in \Delta^*, c \sim c\}$ is injective, and,
(4) letting $\gamma_\sim : \Delta^* \rightharpoonup \Delta^*$ and $\gamma_h : \Delta^* \rightharpoonup \Delta^*$ such that, for each $c \in \Delta^*$

$$\gamma_\sim(c) = \begin{cases} \min([c]_\sim) & \text{if } [c]_\sim \neq \emptyset \\ \text{undefined} & \text{otherwise} \end{cases}$$

$$\gamma_h(c) = \begin{cases} \min(h(c)) & \text{if h(c) is defined} \\ \text{undefined} & \text{otherwise} \end{cases}$$

the execution of $\mathcal{P}$ over the input $(\gamma_\sim(S'), \gamma_\sim(I'), \gamma_\sim(m))$ returns $(\gamma_h(S^{+\prime}), M^+)$, where

$$m = \begin{cases} Q & \text{if } Q \text{ is propositional} \\ Q(\min[q_*]_\sim) & \text{if } Q \text{ is unary} \end{cases}$$

The set T of transitions of N is $T = T' \cup T'''$.

Flow function. The flow function F of N is defined as follows. For each $t = t_{(I,\sim)} \in T'$ and $p \in P$,

$$F(p,t) = \begin{cases} \{\!|x_\emptyset|\!\} & \text{if } p = inp \\ \emptyset & \text{otherwise} \end{cases}$$

$$F(t,p) = \begin{cases} \{\!|x_\emptyset|\!\} & \text{if } p \in \{exe, (in, I')\} \\ \{\!|x_{[p]_\sim}|\!\} & \text{if } p \in \Delta_e \\ \emptyset & \text{otherwise} \end{cases}$$

Given a multiset M of facts and a relational symbol Q, let M/Q denote the *projection of M over Q*, i.e., the multiset of facts such that, for each fact $F(t_1, \ldots, t_n)$,

$$M/Q(F(t_1, \ldots, t_n)) = \begin{cases} M(F(t_1, \ldots, t_n)) & \text{if } F = Q \\ 0 & \text{otherwise} \end{cases}$$

[14] The order allows us to extract the minimum from each subset of Δ^*, favoring Δ_{spec}.

For each $t = t_{(S',I',S^{+\prime},\sim,h,Q)} \in T'''$ and $p \in P$,

$$F(p,t) = \begin{cases} \{\!|x_\emptyset|\!\} & \text{if } p \in \{exe, (st,S'), (in,I')\} \\ \{\!|x_\emptyset|\!\} & \text{if } P = Q \text{ and } Q \text{ is propositional} \\ \{\!|x_{[q_*]_\sim}|\!\} & \text{if } P = Q \text{ and } Q \text{ is unary} \\ \{\!|x_{[p]_\sim}|\!\} & \text{if } p \in \Delta^* \\ \emptyset & \text{otherwise} \end{cases}$$

$$F(t,p) = \begin{cases} \{\!|x_\emptyset|\!\} & \text{if } p \in \{inp, (st,S^{+\prime})\} \\ \{\!|x_{h(p)}|\!\} & \text{if } p \in \Delta_s \\ \{\!|x_\emptyset|\!\} & \text{if } p/0 \in \mathcal{T} \text{ and } p \in M^+ \\ \{\!|x_{h(c_1)}, \ldots, x_{h(c_k)}|\!\} & \text{if } p/1 \in \mathcal{T} \text{ and } M^+/p = \\ & \{\!|p(\min h(c_1)), \ldots, p(\min h(c_k))|\!\} \end{cases}$$

C.2 Encoding

A configuration $C = (S, I, C)$ of D is encoded by a marking M of N if there is some partial function $f : \Delta^* \setminus \{q_*\} \rightharpoonup \Delta$, $S' \in H(\Delta_s, \mathcal{S})$, and $I' \in H(\Delta_e, \mathcal{I})$ such that:

(1) f is defined only on $\Delta(S) \cup \Delta(I) \cup \Delta_{spec}$,
(2) $f|_{\Delta_{spec}} = id|_{\Delta_{spec}}$, $f(S') = S$, and $f(I') = I$,
(3) $M(inp) = \emptyset$ and $M(exe) = \{\!|\bullet|\!\}$,
(4) for each $A \in H(\Delta_s, \mathcal{S}) \cup H(\Delta_e, \mathcal{S})$,

$$M(A) = \begin{cases} \{\!|\bullet|\!\} & \text{if } A \in \{S', I'\} \\ \emptyset & \text{otherwise} \end{cases}$$

(5) for each $c \in \Delta^* \setminus \{q_*\}$,

$$M(c) = \begin{cases} \{\!|f(c)|\!\} & \text{if } f(c) \text{ is defined} \\ \{\!|\bullet|\!\} & \text{otherwise} \end{cases}$$

(6) for each message $Q \in \mathcal{T}$,

$$M(Q) = \begin{cases} \{\!|\underbrace{\bullet, \ldots, \bullet}_{C(Q) \text{ times}}|\!\} & \text{if } Q \text{ is propositional} \\ \sum_{d \in \Delta} \{\!|\underbrace{d, \ldots, d}_{C(Q(d)) \text{ times}}|\!\} & \text{if } Q \text{ is unary} \end{cases}$$

D LANGUAGES

We define the syntax and semantics of $\mathrm{CTL}_{\mathrm{CDP}}$ for single-node CDPs and the syntax of its fragments

CTL_{CDP}. Formulas Φ in $\mathrm{CTL}_{\mathrm{CDP}}$ over a CDP network *Net* are defined by the following grammar:

$$\Phi ::= \varphi \mid \neg\Phi \mid \Phi \wedge \Phi \mid \Phi \vee \Phi \mid \mathsf{EX}\Phi \mid \mathsf{E}(\Phi \,\mathsf{U}\, \Phi) \mid \mathsf{A}(\Phi \,\mathsf{U}\, \Phi)$$
$$\varphi ::= t_1 = t_2 \mid S(\mathbf{t}) \mid T(\mathbf{t}) \mid \exists x.\varphi \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2$$

S is a state symbol, T is a transport symbol, and $\mathbf{t}$ is a tuple of terms.

Given a configuration $C = (\mathbf{S}, \mathbf{I}, \mathbf{T})$ of a b-CDP $\mathcal{D}$ over a network $Net = (V, E, N)$, we inductively define when a formula in $\mathrm{CTL}_{\mathrm{CDP}}$ is true over C under the configuration graph Υ^b:

- $\Upsilon^b, C \models t_1 = t_2$ if $t_1 = t_2$.
- $\Upsilon^b, C \models S(\mathbf{t})$ if $S(\mathbf{t}) \in \mathbf{S}$.

- $\Upsilon^b, C \models T(\mathbf{t})$ if $T(\mathbf{t}) \in \mathit{Supp}(\mathbf{T})$.
- $\Upsilon^b, C \models \exists x.\varphi(x)$ if there is a $c \in \Delta(\mathbf{S}) \cup \Delta(\mathbf{T})$ such that $C \models \varphi(c)$.
- $\Upsilon^b, C \models \neg\varphi$ if $C \models \varphi$ is false.
- $\Upsilon^b, C \models \varphi_1 \wedge \varphi_2$ if $C \models \varphi_1$ and $C \models \varphi_2$.
- $\Upsilon^b, C \models \varphi_1 \vee \varphi_2$ if $C \models \varphi_1$ or $C \models \varphi_2$.
- $\Upsilon^b, C \models \mathsf{EX}\Phi$ if there is a configuration C' such that $C \to C'$ in the configuration graph of $\mathcal{D}$ such that $C' \models \Phi$.
- $\Upsilon^b, C \models \mathsf{E}(\Phi_1 \,\mathsf{U}\, \Phi_2)$ if there is a path $C = C_1 \to \cdots \to C_n$, for some $n \geq 1$, in the configuration graph of $\mathcal{D}$ such that, for each $i \leq n$, $C_i \models \Phi_1$ and $C_n \models \Phi_2$.
- $\Upsilon^b, C \models \mathsf{A}(\Phi_1 \,\mathsf{U}\, \Phi_2)$ if, for each $n \in \mathbb{N}$, for each path $C = C_1 \to \cdots \to C_n$ in the configuration graph of $\mathcal{D}$, for each $i \leq n$, $C_i \models \Phi_1$ and $C_n \models \Phi_2$.

We use the following standard abbreviations.

- boolean abbreviations such as $\top$, $\bot$, $\to$.
- $\forall x.\phi = \neg\exists x.\neg\phi$.
- $\mathsf{AX}\phi = \neg\mathsf{EX}\neg\phi$.
- $\mathsf{EF}\phi = \mathsf{E}(\top \,\mathsf{U}\, \phi)$.
- $\mathsf{AF}\phi = \mathsf{A}(\top \,\mathsf{U}\, \phi)$.
- $\mathsf{EG}\phi = \neg\mathsf{AF}\neg\phi$
- $\mathsf{AG}\phi = \neg\mathsf{EF}\neg\phi$.

The semantics of $\mathsf{EF}[\square, \triangle, st]$ fragments is as that of $\mathrm{CTL}_{\mathrm{CDP}}$, projected over the symbols in the fragment, with the provision that $C \models \exists x.\varphi(x)$ if there is a $c \in \Delta(\mathbf{S})$ such that $C \models \varphi(c)$. The fragments we consider are defined by the following grammars.

$\mathsf{EF}[-, \mathit{pos}, st]$.

$$\begin{array}{rcl} \Phi & ::= & \varphi \mid \mathsf{EF}\varphi \mid \Psi \mid \Phi \wedge \Phi \mid \Phi \vee \Phi \\ \varphi & ::= & t_1 = t_2 \mid S(\mathbf{t}) \mid \exists x.\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \end{array}$$

$\mathsf{EF}[\mathsf{EF}^+, \mathit{pos}, st]$.

$$\begin{array}{rcl} \Phi & ::= & \varphi \mid \mathsf{EF}\varphi \mid \mathsf{EF}\Phi \mid \Psi \mid \Phi \wedge \Phi \mid \Phi \vee \Phi \\ \varphi & ::= & t_1 = t_2 \mid S(\mathbf{t}) \mid \exists x.\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \end{array}$$

$\mathsf{EF}[\mathit{AG}, \mathit{pos}, st]$.

$$\begin{array}{rcl} \Phi & ::= & \varphi \mid \mathsf{EF}\varphi \mid \mathsf{EF}(\varphi \wedge \mathsf{AG}\varphi) \mid \Psi \mid \Phi \wedge \Phi \mid \Phi \vee \Phi \\ \varphi & ::= & t_1 = t_2 \mid S(\mathbf{t}) \mid \exists x.\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \end{array}$$

$\mathsf{EF}[\mathsf{AX}^2, \mathit{pos}, st]$.

$$\begin{array}{rcl} \Phi & ::= & \varphi \mid \mathsf{EF}\varphi \mid \mathsf{EF}(\varphi \wedge \mathsf{AXAX}\varphi) \mid \Psi \mid \Phi \wedge \Phi \mid \Phi \vee \Phi \\ \varphi & ::= & t_1 = t_2 \mid S(\mathbf{t}) \mid \exists x.\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \end{array}$$

For $\square \in \{-, \mathsf{EF}^+, \mathsf{AG}, \mathsf{AX}^2\}$, the fragment $\mathsf{EF}[\square, \mathit{bool}, st]$ is defined as $\mathsf{EF}[\square, \mathit{pos}, st]$ by adding the production rule $\varphi ::= \neg\varphi$; for $\triangle \in \{\mathit{pos}, \mathit{bool}\}$, $\mathsf{EF}[\square, \triangle, st{+}ch]$ is defined as, $\mathsf{EF}[\square, \triangle, st]$ by adding the production rule $\varphi ::= T(\mathbf{t})@\langle n_1, n_2\rangle$.

Received December 2023; revised February 2024; accepted March 2024