Abstract Domains for Database Manipulating Processes

Tobias Schüler[1][0009−0008−1559−133X], Stephan Mennicke[2][0000−0002−3293−2940], and Malte Lochau[1][0000−0002−8404−753X]

[1] University of Siegen, firstname.lastname@uni-siegen.de
[2] Knowledge-Based Systems Group, TU Dresden, Dresden, Germany
stephan.mennicke@tu-dresden.de

Abstract. Database manipulating systems (DMS) formalize operations on relational databases like adding new tuples or deleting existing ones. To ensure sufficient expressiveness for capturing practical database systems, DMS operations incorporate as guarding expressions first-order formulas over countable value domains. Those features impose infinite state, infinitely branching processes thus making automated reasoning about properties like reachability of states intractable. Most recent approaches therefore restrict DMS to obtain decidable fragments. Nevertheless, a comprehensive semantic framework capturing full DMS, yet incorporating effective notions of data abstraction and process equivalence is an open issue. In this paper, we propose DMS process semantics based on principles of abstract interpretation. The concrete domain consists of all valid databases, whereas the abstract domain employs different constructions for unifying sets of databases being semantically equivalent up to particular fragments of the DMS guard language. The connection between abstract and concrete domain is effectively established by homomorphic mappings whose properties and restrictions depend on the expressiveness of the DMS fragment under consideration. We instantiate our framework for canonical DMS fragments and investigate semantical preservation of abstractions up to bisimilarity, being one of the strongest equivalence notions for operational process semantics.

Keywords: database manipulating systems · abstract interpretation · labeled transition systems · bisimulation equivalence.

1 Introduction

Background and Motivation. Modern software systems intensively interact with diverse environmental components which often includes one or more (relational) databases. Database manipulating systems [1] (DMS) and similar approaches [7,8,4,22] characterize the operational behavior of (relational) database systems by formalizing actions consecutively transforming the current state of databases by adding new tuples or deleting existing ones. The action language supported by DMS-like formalisms must be sufficiently expressive to capture

crucial behavioral aspects of practical database systems. To this end, those actions combine set-based add/delete operations with FOL formulas both defined on databases over (countable) value domains [2]. The FOL part serve as guarding expressions for actions which, if enabled, have the ability to further expand and/or narrow the active domain of databases reached in the subsequent state. However, these distinct features of DMS-like formalisms impose intrinsically problematic properties on the underlying operational semantics. For instance, using labeled transitions systems (LTS) [7], the resulting process model is not only non-regular and infinite-state, but even infinitely branching as arbitrary fresh data may be added to databases in a step. Essential correctness properties of DMS processes like reachability of states are thus not only theoretically undecidable, but also practically intractable by state-of-the-art reasoning tools. As a pragmatic workaround, most approaches consider bounded state spaces and/or narrow down expressiveness of DMS languages to obtain decidable fragments [1].

Contributions. In this paper, we apply the framework of *abstract interpretation* [9,10] to tame the LTS semantics of DMS processes. In the concrete domain, the set of LTS states corresponds to all valid databases of a given database schema over infinite value domains. In the abstract domain, LTS states are constructed by employing different abstraction operators for unifying subsets of databases. This abstract representation enables us to effectively connect the abstract and concrete domains by means of homomorphic mappings. The types of properties of DMS processes being preserved and/or reflected by abstraction depend on the expressiveness of the DMS fragment used in DMS actions as well as the notion of process equivalence under consideration. We instantiate our framework for canonical DMS fragments and investigate behavior preservation of abstractions up to bisimilarity. As bisimilarity constitutes one of the strongest equivalence notions for LTS-based process semantics, our abstraction builds the basis for guaranteeing preservation of essential semantical properties. In this way, our framework provides a sound conceptual basis for building effective model-checking tools for DMS process verification [10].

Extended Version. We omit the proofs for the main results due to space restrictions and instead refer to the extended version [24].

2 Foundations

Databases. We assume a first-order (FO) vocabulary consisting of mutually disjoint (countably infinite) sets of constants $\mathbf{C}$, variables $\mathbf{V}$, and predicates $\mathbf{P}$. Each predicate $p \in \mathbf{P}$ has an arity $ar(p) \in \mathbb{N}$. Terms are either constants or variables, and for a list of terms $\mathbf{t} = t_1, \ldots, t_n$ we denote its length by $|\mathbf{t}| = n$. An expression $p(\mathbf{t})$ is an atom if $p \in \mathbf{P}$ and $\mathbf{t}$ is a term list, such that $ar(p) = |\mathbf{t}|$. An atom is *grounded* if it is variable-free and we call a finite set of ground atoms $\mathcal{D}$ a *database*. The universe of all databases is $\mathbb{U}^{\mathbf{C}}$. A (possibly infinite) set of ground atoms $\mathcal{I}$ is an *instance* with the respective universe $\mathbb{I}^{\mathbf{C}}$. Note that, $\mathbb{U}^{\mathbf{C}} \subseteq \mathbb{I}^{\mathbf{C}}$.

Table 1. Guard fragments, their formula shape, and their abbreviation

guard fragment	abbrv.	formula
normal conjuncitve guard	NCG	$\exists \mathbf{y}.\ a_1 \wedge \ldots \wedge a_m \wedge \neg b_1 \wedge \ldots \wedge \neg b_n$
projection-free NCG	pf-NCG	$a_1 \wedge \ldots \wedge a_m \wedge \neg b_1 \wedge \ldots \wedge \neg b_n$
conjuncitve guard	CG	$\exists \mathbf{y}.\ a_1 \wedge \ldots \wedge a_m$
projection-free CG	pf-CG	$a_1 \wedge \ldots \wedge a_m$
conjunction of negated atoms	CNA	$\forall \mathbf{y}.\ \neg a_1 \wedge \ldots \wedge \neg a_m$

Guards. We consider FOL formulas g to serve as guards as follows:

$$\Phi ::= p(\mathbf{t}) \mid t = u \mid \neg\Phi \mid \Phi \wedge \Phi \mid \exists x.\ \Phi \tag{1}$$

where $p(\mathbf{t})$ is an atom, t, u are terms, and $x \in \mathbf{V}$. The terms occurring in guard g being variables are referred to by the set $\mathit{vars}(g)$. A variable $x \in \mathit{vars}(g)$ is either *free* or *bound* in g, defining the set $\mathit{free}(g)$[3] of free variables.

Guards as FOL Fragments. A *normal conjunctive guard* (NCG) is a formula

$$\exists \mathbf{y}.\ a_1 \wedge \ldots \wedge a_m \wedge \neg b_1 \wedge \ldots \wedge \neg b_n \tag{2}$$

where $\mathbf{y}$ is a list of variables occurring in the atoms $a_1, \ldots, a_m, b_1, \ldots, b_n$. For an NCG g of shape (2) we refer to the positive guard part by $g^+ = \exists \mathbf{y}.\ a_1 \wedge \ldots \wedge a_m$ and its negated part by $g^- = \exists \mathbf{y}.\ \neg b_1 \wedge \ldots \wedge \neg b_n$, respectively. Whenever convenient, g^+ (g^-, resp.) identifies the set of atoms occurring within g, meaning $g^+ = \{a_1, \ldots, a_m\}$ ($g^- = \{b_1, \ldots, b_n\}$, resp.). An NCG g with $g^- = \emptyset$ is a *conjunctive guard* (CG). An NCG g is *safe* if $\mathit{vars}(g^-) \subseteq \mathit{vars}(g^+)$. Similarly, the other guard fragments are summarized in Table 1.

Substitution. A *substitution* is a partial function $\sigma : \mathbf{V} \to \mathbf{C}$ mapping variables to constants. The set of all variables for which σ is defined is denoted by $\mathit{dom}(\sigma)$. We call σ a *substitution for guard* g if $\mathit{vars}(g) \subseteq \mathit{dom}(\sigma)$. Such a substitution replaces variables of a guard by constants and, thereby, forms a *guard match*. For convenience, we assume for every substitution σ and constant $c \in \mathbf{C}$, $\sigma(c) = c$, extending the signature of σ to $\mathbf{V} \cup \mathbf{C} \to \mathbf{C}$. If $\mathbf{t} = t_1 \ldots t_n$ is a list of terms and σ a substitution defined for all variables in $\mathbf{t}$, we denote by $\mathbf{t}\sigma$ the term list $\sigma(t_1) \ldots \sigma(t_n)$. A substitution σ is a *match to guard* g *in instance* $\mathcal{I}$ if (a) $\mathit{free}(g) = \mathit{dom}(\sigma)$ and (b) $\mathcal{I}, \sigma \models g$, where

- $\mathcal{I}, \sigma \models p(\mathbf{t})$ if $p(\mathbf{t}\sigma) \in \mathcal{I}$,
- $\mathcal{I}, \sigma \models t = u$ if $t\sigma = u\sigma$,
- $\mathcal{I}, \sigma \models \neg g$ if $\mathcal{I}, \sigma \models g$ does not hold,
- $\mathcal{I}, \sigma \models g \wedge g'$ if $\mathcal{I}, \sigma \models g$ and $\mathcal{I}, \sigma \models g'$, and
- $\mathcal{I}, \sigma \models \exists x.\ g$ if $\mathcal{I}, \sigma[x \mapsto c] \models g$ for some $c \in \mathbf{C}$.

[3] $\mathit{free}(r(\mathbf{t})) = \mathbf{t} \cap \mathbf{V}$, $\mathit{free}(t = u) = \{t, u\} \cap \mathbf{V}$, $\mathit{free}(\neg g) = \mathit{free}(g)$, $\mathit{free}(g \wedge \psi) = \mathit{free}(g) \cup \mathit{free}(\psi)$, and $\mathit{free}(\exists x.\ g) = \mathit{free}(g) \setminus \{x\}$.

Guard matches. We denote the set of all matches to guard g in instance $\mathcal{I}$ by $g(\mathcal{I})$. We may simply write g to identify a guard. A guard match to NCGs $g = \exists \mathbf{y}.\ \psi$ in $\mathcal{I}$ is tightly connected to the existence of homomorphisms from ψ (viewed as a set of atoms) to instance $\mathcal{I}$. A function $h : \mathbf{C} \cup \mathbf{V} \to \mathbf{C} \cup \mathbf{V}$ is called a *homomorphism* from a set of atoms $\mathcal{A}$ into a set of atoms $\mathcal{B}$ if (a) $h(c) = c$ for all $c \in \mathbf{C}$ and (b) $p(t_1, \ldots, t_n) \in \mathcal{A}$ implies $p(h(t_1), \ldots, h(t_n)) \in \mathcal{B}$.

Guard match, query answer and substitution. Guards and guard matches are very similar to queries and query answers in database systems. However, whereas query answers should be *domain independent* (i.e., having finitely many possible substitutions [2]), this does not necessarily hold for guard matches [1]. For instance, query $\neg P(x)$ would have infinitely many answers and is therefore prohibited, whereas the corresponding guard simply checks if, for instance, a to-be-added person is not yet contained in the database.

Example 1. We consider a simplified social network (SSN) with two predicates, (1) a unary predicate $P(\mathit{name})$ for *persons* currently being members of the network with attributes *name*, and (2) a binary predicate $F(\mathit{name}_1, \mathit{name}_2)$ for a non-symmetric *friendship* relation from person name_1 to person name_2. We assume all possible strings denoting names to be part of $\mathbf{C}$. A database of our SSN may be $\mathcal{D}_e = \{P(\mathrm{A}), P(\mathrm{B}), P(\mathrm{C}), F(\mathrm{A}, \mathrm{B}), F(\mathrm{B}, \mathrm{A}), F(\mathrm{A}, \mathrm{C})\}$ (with A, B and C may be Alice, Bob and Charles). Potential guards are

- a symmetric friendship: $g_{sf} = F(x, y) \wedge F(y, x)$,
- a directed friendship: $g_{df} = F(x, y) \wedge \neg F(y, x)$,
- a friendship from x to someone: $g_{af} = \exists y.F(x, y)$, and
- no friendship: $g_{nf} = \neg F(x, y) \wedge \neg F(y, x)$.

On $\mathcal{D}_e$ we obtain the following guard matches:

- $g_{sf}(\mathcal{D}_e) = \{\{x \mapsto \mathrm{A}, y \mapsto \mathrm{B}\}, \{x \mapsto \mathrm{B}, y \mapsto \mathrm{A}\}\}$,
- $g_{df}(\mathcal{D}_e) = \{\{x \mapsto \mathrm{A}, y \mapsto \mathrm{C}\}\}$,
- $g_{sf}(\mathcal{D}_e) = \{\{x \mapsto \mathrm{A}\}, \{x \mapsto \mathrm{B}\}\}$, and
- $g_{nf}(\mathcal{D}_e) = \{\{x \mapsto \mathrm{B}, y \mapsto \mathrm{C}\}, \{x \mapsto \mathrm{C}, y \mapsto \mathrm{B}\}, \{x \mapsto \mathrm{A}, y \mapsto \mathrm{A}\}, \ldots\}$.

For instance $\mathcal{D}_e, \{x \mapsto \mathrm{A}, y \mapsto \mathrm{B}\} \models g_{sf}$ holds as Alice is a friend of Bob and Bob is a friend of Alice, whereas $\mathcal{D}_e, \{x \mapsto \mathrm{A}, y \mapsto \mathrm{B}\} \models g_{df}$ does not hold.

Database Manipulating Systems. Database manipulating systems formalize possible sequences of *actions* consecutively applied to database instances. Syntactically, our formalization loosely follows the canonical notion of actions used in the DMS formalism by Abdulla et al. [1]. An action consists of a *guard* and an *effect* on the current instance. A guard specifies on which instances the action is applicable. The effect might be deletion of atoms from the instance and adding new atoms to the instance. Formally, the effect comprises two finite sets of atoms, `Del` and `Add`, such that $\mathit{vars}(\texttt{Del}) \subseteq \mathit{free}(\texttt{g})$. Atoms in `Del` are determined by the match for guard g, while `Add` is a collection of new atoms. Note that `Del` and `Add` may contain variables that will be bound by (a) the guard matches and (b) by

arbitrary constants in case of those variables in $vars(\texttt{Add}) \setminus free(\texttt{g})$. The rationale behind case (b) is that an action inserting atoms may depend on external stimuli like sensor data or user input. An action *act* is a triple $(g, \texttt{Del}, \texttt{Add})$ which forms the basis of a *database manipulating system* (DMS).

Definition 1 (Database Manipulating System). *A* database manipulating system *(DMS) is a pair* $\mathcal{S} = (\mathcal{I}_0, \textsc{Act})$ *where* $\mathcal{I}_0$ *is the* initial instance *and* Act *is a finite set of actions.*

From $\mathcal{I}_0$, any sequence of actions $act = (g, \texttt{Del}, \texttt{Add}) \in \textsc{Act}$ may be performed based on substitutions σ due to matches of guard g. Note that σ specifies all variables occurring in Del. We denote by $\texttt{Del}\sigma$ the set obtained by replacing all occurrences of variables $x \in vars(\texttt{Del})$ by $\sigma(x)$. In general, for a set of atoms $\mathcal{A}$ and substitution σ, $\mathcal{A}\sigma$ is the set of atoms in which each variable $x \in vars(\mathcal{A})$ has been replaced by $\sigma(x)$ if it is defined for σ. Set Add may contain variables which are not in $dom(\sigma)$ such that $\texttt{Add}\sigma$ is not a proper database . To facilitate arbitrary external inputs, we expand σ to the missing variables. Substitution $\sigma^\star$ extends σ to Add if $dom(\sigma^\star) = \texttt{Add}$ and $\sigma \subseteq \sigma^\star$. Extending σ to $\sigma^\star$ completes a step by deletions $\texttt{Del}\sigma^\star$ from and additions $\texttt{Add}\sigma^\star$ to the current instance.

Definition 2 (DMS Step). *A DMS action* $act = (g, \texttt{Del}, \texttt{Add})$ *is* enabled under instance $\mathcal{I}$ and substitution σ*, denoted* $\mathcal{I}[act, \sigma\rangle$*, if* $\sigma \in g(\mathcal{I})$*. If* $\mathcal{I}[act, \sigma\rangle$*, then an effect is an extension* $\sigma^\star$ *of* σ *to* Add*, producing instance* $\mathcal{I}' = (\mathcal{I} \setminus \texttt{Del}\sigma^\star) \cup \texttt{Add}\sigma^\star$*. We denote the* DMS step *from* $\mathcal{I}$ *to* $\mathcal{I}'$ *via act and* σ *by* $\mathcal{I}\ [act, \sigma^\star\rangle\ \mathcal{I}'$*.*

Example 2. Action $act_{add} = (true, \emptyset, \{P(x)\})$ (adding a new person) is enabled under each instance even if the person already exists in that instance. Thus, σ is empty and $\sigma^\star$ may be, e.g., $\{x \mapsto \mathrm{A}\}$. Action $act_{rev} = (g_{df}, \{F(x,y)\}, \{F(y,x)\})$ checks if a directed friendship exists between x and y, deletes this friendship and adds the reversed friendship.

The formal semantics of a DMS is defined as a *labeled transition system* (LTS).

Definition 3 (Labeled Transition System). *A* labeled transition system *(LTS) is a triple* $\mathcal{T} = (Q, \Sigma, \Rightarrow)$ *where* q *is a set of states (processes),* Σ *is a set of transition labels, and* $\Rightarrow \subseteq Q \times \Sigma \times Q$ *a transition relation. We denote* $(q, a, q') \in \Rightarrow$ *as* $q \overset{a}{\Rightarrow} q'$ *and write* $q \overset{a}{\Rightarrow}$ *if* $\exists q' \in Q : q \overset{a}{\Rightarrow} q'$ *and* $q \overset{a}{\not\Rightarrow}$ *if not* $q \overset{a}{\Rightarrow}$*.*

LTS $\mathcal{T} = (Q, \Sigma, \Rightarrow)$ is (a) *finitely branching* if for each $q \in Q$, the set $\{q' \in Q \mid \exists a \in \Sigma : q \overset{a}{\Rightarrow} q'\}$ is finite, (b) *image-finite* if for each $q \in Q$ and $a \in \Sigma$, the set $\{q' \in Q \mid q \overset{a}{\Rightarrow} q'\}$ is finite, (c) *finite-state* if Q is finite, and (d) *deterministic* if for each state $q \in Q$ and $a \in \Sigma$, $q \overset{a}{\Rightarrow} q'$ and $q \overset{a}{\Rightarrow} q''$ implies $q' = q''$. Although LTSs may be directly associated with directed edge-labeled graphs, comparison relations based on graph homomorphisms are too strong to capture distinctive features of LTS processes. Instead, *simulation* and *bisimulation* relations on processes are used. Intuitively, process q simulates p if every action that may be performed by p can be mimicked by q and the successor states again simulate each other.

Definition 4 ((Bi-)Simulation). *For an LTS* $(Q, \Sigma, \Rightarrow)$*, a binary relation* $R \subseteq Q \times Q$ *is a* simulation *if for all* $(p, q) \in R$ *and* $a \in \Sigma$*,* $p \overset{a}{\Rightarrow} p'$ *implies that* $q' \in Q$ *exists such that* $q \overset{a}{\Rightarrow} q'$ *and* $(p', q') \in R$*. Process* $q \in Q$ simulates *process* $p \in Q$ *if there is a simulation* R *with* $(p, q) \in R$*. If* p *simulates* q *by simulation* R*, and* q *simulates* p *by simulation* R'*, then* p *and* q *are* similar*. Simulation* R *is a* bisimulation *if, and only if,* $R^{-1} := \{(q, p) \mid (p, q) \in R\}$ *is also a simulation. If there is a bisimulation* R*, such that* $(p, q) \in R$*, then* p *and* q *are* bisimilar*.*

Note, the witnesses R and R' for similarity are not necessarily bisimulations as possibly $R^{-1} \neq R'$.

DMS semantics can be formalized as an LTS $\mathsf{DMS} := (\mathbb{U}^{\mathbf{C}}, \textsc{Act}\Sigma, \Rightarrow)$ where $\Rightarrow$ is formed by $\mathcal{I}_1 \overset{\langle act, \sigma \rangle}{\Longrightarrow} \mathcal{I}_2$ if, and only if, $\mathcal{I}_1 \; [act, \sigma\rangle \; \mathcal{I}_2$ (cf. Def. 2). In general, DMS is infinitely branching, infinite-state, and deterministic.

DMS builds the basis for investigating desirable properties of all possible processes defining a DMS. For instance, the *reachability problem* asks for a given DMS and a distinguished action act_x, if there is an instance $\mathcal{I}_x$ with $\mathcal{I}_0 \Rightarrow \mathcal{I}_1 \Rightarrow \ldots \Rightarrow \mathcal{I}_x$ such that action act_x is enabled under $\mathcal{I}_x$. The reachability problem is undecidable for DMS [1]. The next example constitutes a semi-decidable reachability problem.

Example 3. Given a predefined set of persons and actions for consecutively adding and deleting friendships between arbitrary pairs of persons, do we eventually reach a database containing a triangle friendship between three different persons (i.e., x a friend of y, y a friend of z and z a friend of x)? To this end, we expand the unary predicate $P(name)$ to a binary predicate $P(name, name)$ and use NCG to define a guard $P(x, x) \wedge P(y, y) \wedge \neg P(x, y).\, \neg P(x, y)$ (i.e., ensuring that x and y match different persons). This is a standard technique to avoid $\neq$ in first order formulas. Starting from an arbitrary database, we consider two actions: $act_{add} := (P(x, x) \wedge P(y, y) \wedge \neg P(x, y), \emptyset, F(x, y))$ (adding a friendship) and $act_{delete} := (F(x, y), F(x, y), \emptyset)$ (deleting a friendship) and ask for reachability of the action $act_{end} = (\exists x, y, z.F(x, y) \wedge F(y, z) \wedge F(z, x) \wedge \neg P(x, y) \wedge \neg P(y, z) \wedge \neg P(z, x), \emptyset, \emptyset)$.

A finite solution to this problem comprises an *abstract LTS* with four states, where each of those abstract states contains all subsets of databases with (1) no friendships, (2) friendship chains of maximum length ≤ 2, (3) friendship chains of maximum length > 2 without any triangles, and (4) at least one triangle.

In the remainder of this paper, we develop a hierarchy of abstract domains to characterize semantic-preserving abstractions of states of DMS depending on the expressiveness of the guard fragment used. Our approach is based on the formal framework of abstract interpretation.

3 Principles of Abstract Interpretation

Before we present our abstract interpretation framework for DMS, we first describe its basic ingredients. Different processes assembled in DMS may share

similar behavior in terms of their enabled actions and subsequent processes. For instance, let us consider a DMS action which inserts a friendship between Alice and Bob, where the guard of this action consists of a conjunction of atoms requiring Alice and Bob to exist in the database. All (i.e., countably infinitely many) *concrete states* matching this guard may be aggregated into *one* single abstract state. The concrete states aggregated in the subsequent abstract state reached after performing this action then all share the inserted relationship between Alice and Bob. The way how the concrete states are aggregated into, and reconstruction from, such an abstract state clearly depends on the guard fragment used. In addition, DMS states are infinitely branching due to the ability of DMS actions to insert any possible new value. However, in many cases, the exact values are often not relevant for reasoning about the subsequent behavior and can therefore be aggregated into one representative abstract value. The following definitions are based on Dams et al. [10] and conceptualize these observations.

Lattice. Abstract interpretation provides a framework for effectively reasoning about computational models over infinite semantic domains modeled as lattices. By $\sqcap$ and $\sqcup$ we denote binary operations on sets S. The operators $\sqcap$ and $\sqcup$ are monotone with respect to a partial order $\leq$ on S (i.e., $x_1, x_2, y_1, y_2 \in S$, $x_1 \leq x_2$ and $y_1 \leq y_2$ implies $x_1 \sqcap y_1 \leq x_2 \sqcap y_2$ and $x_1 \sqcup y_1 \leq x_2 \sqcup y_2$).

Definition 5 (Lattice). *A* lattice *is a partially ordered set $(S, \leq)$ such that each two-element subset $\{x, y\} \subseteq S$ has (1) a unique least upper bound in S, denoted by $x \sqcup y$, and (2) a unique greatest lower bound in S, denoted by $x \sqcap y$.*

Bounded lattices are not further deployed in the following but are mentioned here only for the sake of comprehensibility. Abstract interpretation aims at establishing connections between lattices modeling different semantic domains.

Galois Connection. By $(\mathbb{C}, \sqsubseteq)$ we denote a concrete semantic domain where $\mathbb{C} = 2^Q$ comprises the set of all subsets of concrete sets of states Q of a computational model (here: DMS). By $\sqsubseteq$ we denotes a partial (semantic) ordering on $\mathbb{C}$ (here: $\subseteq$). By $(\mathbb{A}, \preceq)$ we denote an abstract semantic domain where $\mathbb{A}$ is a set of abstract states and $\preceq$ a partial (precision) ordering on $\mathbb{A}$. It is crucial that the elements of the concrete domain $\mathbb{C}$ are possible *subsets* of concrete states, whereas the elements of the abstract domain $\mathbb{A}$ are *singleton* abstract states. The mutual connection between concrete and abstract domain is shaped by a pair of abstraction function $\alpha : \mathbb{C} \to \mathbb{A}$ and a concretization function $\gamma : \mathbb{A} \to \mathbb{C}$, together forming a *Galois connection.*

Definition 6 (Galois Connection). *The pair $(\alpha : \mathbb{C} \to \mathbb{A}, \gamma : \mathbb{A} \to \mathbb{C})$ is a* Galois connection *between lattices $(\mathbb{C}, \sqsubseteq)$ and $(\mathbb{A}, \preceq)$ if (1) α and γ are* total *and* monotone*, (2) $\forall C \in \mathbb{C} : \gamma \circ \alpha(C) \sqsupseteq C$, and (3) $\forall a \in \mathbb{A} : \alpha \circ \gamma(a) \preceq a$.*

Monotonicity guarantees that more precise abstractions single out fewer concrete states and, conversely, abstracting larger sets of concrete states yields less precise abstractions. Furthermore, (2) requires that concrete states are preserved after reconstruction. Finally, (3) requires a form of optimality of the abstraction thus not decreasing precision.

Bisimulation. Lifting (bi-)simulations to steps $C \overset{a}{\Rightarrow} C'$ between sets of concrete states, as apparent in the concrete domain, amounts to all databases $q \in C$ evolving to $q' \in C'$ via $q \overset{a}{\Rightarrow} q'$. We refer to these as $\forall$-steps and adapt the notions of (bi-)simulations accordingly.

Definition 7 ($\forall$-(bi-)simulation). *For abstract domain $(\mathbb{A}, \preceq)$ and concrete domain $(\mathbb{C}, \sqsubseteq)$, a binary relation $R \subseteq \mathbb{A} \times \mathbb{C}$ is a* $\forall$-simulation *if for all $(\mathcal{A}, C) \in R$ and $a \in$ Act, $\mathcal{A} \overset{a}{\Rightarrow} \mathcal{A}'$ implies that there is a $C' \in \mathbb{C}$ such that (a) $C \overset{a}{\Rightarrow} C'$ with a $q \in C$ for each $q' \in C'$ such that $q \overset{a}{\Rightarrow} q'$, and (b) $(\mathcal{A}', C') \in R$ and (c) for each $q \in C$ there is a $q' \in C'$ with $q \overset{a}{\Rightarrow} q'$.*

If $(\mathcal{A}, C) \in R$ and R is a $\forall$-simulation, we say that C $\forall$-simulates $\mathcal{A}$. By reversing the conditions of $\forall$-simulations, we get $\forall$-simulations between the concrete domain and the abstract domain (i.e., $R \subseteq \mathbb{C} \times \mathbb{A}$). Naturally, a $\forall$-simulation R is called a $\forall$-bisimulation if, and only if, R^{-1} is a $\forall$-simulation. We call $\mathcal{A}$ and C $\forall$-bisimilar if, and only if, a $\forall$-bisimulation between $\mathcal{A}$ and C exists.

Abstract Interpretation Framework. The remainder of this paper is devoted to a hierarchy of concrete domains $(\mathbf{2}^{\mathbb{U}}, \subseteq)$ for DMS processes shaped by different fragments of FOL as guard language, where the functions γ and α are either based on the supremum or infimum of the corresponding abstract domains. For a guard language $\mathcal{L}$, we call a Galois connection (α, γ) an *abstract interpretation w.r.t.* $\mathcal{L}$ if for each set of databases $C \subseteq \mathbb{U}$ and set of DMS actions Act using only guards from $\mathcal{L}$, $\alpha(C)$ and C are $\forall$-bisimilar.

4 Abstract Interpretation of DMS

The concrete domain is fixed: $(\mathbf{2}^{\mathbb{U}}, \subseteq)$. For (possibly infinite) sets C of databases, we effectively present six different abstractions: The first two very basic ones are based on (set) union and intersection. The third abstraction is a (Cartesian) combination of the two prior abstractions with the benefit of supporting a more practical guard fragment. One caveat about these abstractions is that we have to waive projection (i.e., existential quantification). To gain DMS actions with more expressive guards, and thereby capture more realistic systems, we devise abstractions for more general abstract domains. We expand our abstract domain incorporating so-called *labeled nulls* as terms in abstract instances. The order on the abstract domain is then based on homomorphisms. The three remaining abstractions are complements of the first three, now in the more abstract domain incorporating labeled nulls. Table 2 summarizes our results.

The rest of this section is structured as follows. First, we introduce a naive set-based abstraction based on the set union operator on databases together with a summary of further set-based abstractions. Resolving the issue of neglecting variable projections in guards we introduce instances with labeled nulls, on which CGs can be used without losing precision. Finally, we combine unions and intersections to even support DMS actions with NCGs. Other abstractions are mentioned in results only, whereas our extended paper provides further explanations and proofs [24].

Table 2. Abstract Domains, Interpretations, and Respective Guard Fragments

abstract domain	$\alpha(C)$	$\gamma(\mathcal{I})$	fragment	
$(\mathbb{I}, \subseteq)$	$\bigcup C$	$\{\mathcal{D} \subseteq \mathcal{I}\}$	CNA	Theorem 1
$(\mathbb{I}, \supseteq)$	$\bigcap C$	$\{\mathcal{I} \subseteq \mathcal{D}\}$	pf-CG	Theorem 2
$(\mathbb{I} \times \mathbb{I}, \leq)$	$(\bigcap C, \bigcup C)$	$\{\mathcal{I} \subseteq \mathcal{D} \wedge \mathcal{D} \subseteq \mathcal{I}\}$	pf-NCG	Theorem 3
$(\mathbb{I}^{\mathbf{N}}, \rightarrow)$	$\bigsqcup C$	$\{\mathcal{D} \rightarrow \mathcal{I}\}$	CNA	Theorem 5
$(\mathbb{I}^{\mathbf{N}}, \leftarrow)$	$\bigsqcap C$	$\{\mathcal{I} \rightarrow \mathcal{D}\}$	CG	Theorem 4
$(\mathbb{I}^{\mathbf{N}} \times, \mathbb{I}^{\mathbf{N}}, \preceq)$	$(\bigsqcap C, \bigsqcup C)$	$\{\mathcal{I} \rightarrow \mathcal{D} \wedge \mathcal{D} \rightarrow \mathcal{I}\}$	NCG	Theorem 6

4.1 Set-Based Abstractions: The Case of Union

As a first and very basic abstraction we study $\bigcup C$ of any set $C \in \mathbf{2}^{\mathbb{U}}$ of databases. If C is infinite, $\bigcup C$ is infinite as well, meaning that $\bigcup C$ is captured in $\mathbb{I}$. Henceforth, we facilitate $\bigcup C$ via the abstraction function $\alpha_1 : \mathbf{2}^{\mathbb{U}} \rightarrow \mathbb{I}$ with $\alpha_1(C)$.

$$\alpha_1(C) := \bigcup C \qquad \gamma_1(\mathcal{I}) := \{\mathcal{D} \subseteq \mathcal{I} \mid \mathcal{D} \text{ is a database}\} \tag{3}$$

The natural choice for the abstract domain is, thus, $(\mathbb{I}, \subseteq)$ because the more databases C contains, the bigger the abstract instance is (cf. Def. 6 item 1). The counterpart concretization function $\gamma_1 : \mathbb{I} \rightarrow \mathbf{2}^{\mathbb{U}}$ is determined by α_1: While α_1 forms the union of all databases contained in a set of databases C, an abstract instance then describes all databases that are (finite) subsets of the abstract instance. $\gamma_1(\mathcal{I})$ is defined in (3).

Databases are finite by definition, implying that if $\mathcal{I}$ is infinite, $\mathcal{D} \subsetneq \mathcal{I}$ for every $\mathcal{D} \in \gamma_1(\mathcal{I})$. The functions in (3) make up for a Galois connection.

Proposition 1. *(α_1, γ_1) is a Galois connection.*

For $C \in \mathbf{2}^{\mathbb{U}}$, we are interested in the behavioral properties of the abstraction $\alpha_1(C)$. Therefore, observe that for every database $\mathcal{D} \in C$, $\mathcal{D} \subseteq \alpha_1(C)$. Thus, guards asking for the absence of atoms will have the same matches on all the databases in C as well as the abstraction $\alpha_1(C)$.

Example 4. We analyze two guards g_{nf} (absence of a friendship) and g_{sf} (presence of a symmetric friendship) from example 1 on C and $\mathcal{I}$ with $C = \{\{P(\mathrm{A}), P(\mathrm{B}), F(\mathrm{A},\mathrm{B}), F(\mathrm{B},\mathrm{A})\}, \{P(\mathrm{A}), P(\mathrm{B}), P(\mathrm{C})\}\}$ and $\mathcal{I} = \bigcup C = \{P(\mathrm{A}), P(\mathrm{B}), P(\mathrm{C}), F(\mathrm{A},\mathrm{B}), F(\mathrm{B},\mathrm{A})\}$. If a friendship is absent in each database of C, this friendship is also absence in $\mathcal{I}$ (i.e., the union of all databases of C). If a friendship is absent in $\mathcal{I}$ this friendship is also absent in each database of C. In contrast, the presence of a symmetric friendship like $F(\mathrm{A},\mathrm{B}), F(\mathrm{B},\mathrm{A})$ holds for $\mathcal{I}$ but not for each database in C.

The guard $g_{nf} = \forall y.\neg F(x, y)$ ensures the absence of all friendships of a person x through the universal quantifier. g_{nf} behaves similar to g_{nf}. The behavior of the existential quantifier is conversely. For instance, $g_{ex} = \exists x.\neg P(x)$ holds for each database as databases are finite but the set of all constants is infinite. In contrast, if set C is infinite and for each constant c, $P(c)$ is contained in some database in C, $\mathcal{I} = \bigcup C$ does not satisfy g_{ex}.

As the examples show, $\alpha_1(C)$ may enable DMS actions with conjunctive guards that are not enabled by some, or any, of the concrete databases in C. Thus, $\alpha_1(C)$ captures the behavior of all databases in C if we choose CNA guards.

Theorem 1. *(α_1, γ_1) is an abstract interpretation w.r.t. CNA guards.*

Similarly, we obtain an abstraction framework based on intersection of all the databases contained in set C of concrete databases.

Theorem 2. *Galois connection (α_2, γ_2) with $\alpha_2(C) := \bigcap C$ and $\gamma_2(\mathcal{I}) := \{\mathcal{D} \in \mathbb{U} \mid \mathcal{I} \subseteq \mathcal{D}\}$ is an abstract interpretation for pf-CGs.*

This is a special case of Theorem 4 (cf. next subsection). Furthermore, combining both former abstractions allows us to cover projection-free normal conjunctive guards in DMS actions. The rationale behind this abstraction is that for an NCG g, g^+ is evaluated on the intersection component while g^- is simultaneaously evaluated on the union component of the abstraction.

Theorem 3. *For $\alpha_3(C) := (\alpha_1(C), \alpha_2(C))$ and $\gamma_3((\mathcal{I}^\cup, \mathcal{I}^\cap)) := \{\mathcal{D} \in \mathbb{U} \mid \mathcal{I}^\cap \subseteq \mathcal{D} \subseteq \mathcal{I}^\cup\}$, Galois connection (α_3, γ_3) is an abstract interpretation for pf-NCGs.*

Next, we consider abstractions allowing for projections (i.e., existentially quantified variables in DMS action guards) to fully capture NCGs in DMS actions.

4.2 Abstractions with Labeled Nulls: The Case of Intersection

There are two issues with the abstractions discussed so far: (a) limited expressiveness in guards of DMS actions (no existential quantification) and (b) (still) infinite branching of abstract states. The reason for the latter is that abstract instances resemble their concrete counterparts too explicitly. To resolve both issues we use the well-known *labeled null* abstraction to get a notion of existence of values contained in a database whose exact values are irrelevant. Finite branching is a welcome side-effect of this abstraction as well as a precise abstraction for DMSs using CGs (including projection via existential quantification).

Labeled nulls are introduced in our framework as a countably infinite set $\mathbf{N}$ (disjoint from all other term sets). As labeled nulls are proxies for the existence of values (i.e., constants), a database, in which every occurrence of a null is replaced by a constant (or other null), is certainly related to the instance that uses the null. Let us denote the set of all instances using constants and labeled nulls by $\mathbb{I}^{\mathbf{N}}$ (short for $\mathbb{I}_{\mathbf{P}}^{\mathbf{C} \cup \mathbf{N}}$). The notions of homomorphisms and guard matches naturally extend to databases containing nulls (i.e., constants must still map to constants, but nulls may map to nulls or constants).

Due to the nature of labeled nulls, their identity does not have the same role as constants have. It is natural to consider $\mathbb{I}^{\mathbf{N}}$ closed under equivalence up to homomorphisms. This means, instances $\mathcal{I}, \mathcal{J} \in \mathbb{I}^{\mathbf{N}}$ are equal, denoted $\mathcal{I} \leftrightarrows \mathcal{J}$, if $\mathcal{I} \to \mathcal{J}$ and $\mathcal{J} \to \mathcal{I}$. Note, on $\mathbb{U}$ equivalence up to homomorphisms coincides with set equality. For instance $\{P(\mathrm{A})\} \leftrightarrows \{P(\mathrm{A}), P(\mathbf{n}_0)\}$ because we can map A

on A and $\mathbf{n}_0$ on A. $\{F(\mathbf{n}_0, \mathbf{n}_1)\} \to \{F(\mathbf{n}_0, \mathbf{n}_0)\}$ but $\{F(\mathbf{n}_0, \mathbf{n}_0)\} \not\to \{F(\mathbf{n}_0, \mathbf{n}_1)\}$ because we can not map $\mathbf{n}_0$ on $\mathbf{n}_0$ and $\mathbf{n}_0$ on $\mathbf{n}_1$.

$(\mathbb{I}^{\mathbf{N}}, \to)$ forms a lattice and, by duality, $(\mathbb{I}^{\mathbf{N}}, \leftarrow)$, too. The join $\sqcup$ of $(\mathbb{I}^{\mathbf{N}}, \to)$ is simply the union of the instances. Conversely, $\sqcap$ is an intersection of two instances generalizing common atoms with different constants via null assertions. For instance, $\mathcal{I} = \{P(\mathrm{A}), P(\mathrm{B}), F(\mathbf{n}_0, \mathbf{n}_1)\}$ and $\mathcal{J} = \{P(\mathrm{A}), P(\mathrm{C}), F(\mathrm{A}, \mathrm{C})\}$ have $\mathcal{I} \sqcup \mathcal{J} = \{P(\mathrm{A}), P(\mathrm{B}), P(\mathrm{C}), F(\mathbf{n}_0, \mathbf{n}_1), F(\mathrm{A}, \mathrm{C})\}$ as least upper bound and the greatest lower bound is $\mathcal{I} \sqcap \mathcal{J} = \{P(\mathrm{A}), F(\mathbf{n}_0, \mathbf{n}_1)\}$.

The next two definitions describe how an action is performed in $(\mathbb{I}^{\mathbf{N}}, \to)$. Let $\mathcal{I}$ be an instance and $act = (g, \mathtt{Del}, \mathtt{Add})$ a DMS action. Instead of extending guard matches σ to $\sigma^\star$ (involving some constants that are added to the instance through variables in $\mathtt{Add}$), we consider extensions of σ that insert (globally) fresh labeled nulls for all variables in $\mathit{vars}(\mathtt{Add}) \setminus \mathit{free}(\mathtt{g})$.

Definition 8. *Let $act = (g, \mathtt{Del}, \mathtt{Add})$ be a DMS action. For abstract instance $\mathcal{I} \in \mathbb{I}^{\mathbf{N}}$, if $\sigma \in g(\mathcal{I})$, then $\mathcal{I} \xRightarrow{\langle act, \sigma \rangle} (\mathcal{I} \setminus \mathtt{Del}\sigma^\star) \cup \mathtt{Add}\sigma^\star$ where $\sigma \subseteq \sigma^\star$ and for each variable $x \in \mathit{vars}(\mathtt{Add}) \setminus \mathit{free}(\mathtt{g})$, $\sigma^\star(x)$ is a fresh labeled null.*

Example 5. For action $act_{add} = (\mathit{true}, \emptyset, \{P(x)\})$ from example 2, $\mathit{vars}(\mathtt{Add}) \setminus \mathit{free}(\mathtt{g}) = \{x\} \setminus \emptyset = \{x\}$ and $\sigma^\star(x) = \mathbf{n}$. We obtain $\emptyset \xRightarrow{\langle act_{add}, \emptyset \rangle} \{P(\mathbf{n})\}$.

Note that the action label only contains the match σ and not its extension. The reason is that for instances $\mathcal{I}, \mathcal{B}_1, \mathcal{B}_2$ and action-match pair $\langle act, \sigma \rangle$, if $\mathcal{I} \xRightarrow{\langle act, \sigma \rangle} \mathcal{B}_1$ and $\mathcal{I} \xRightarrow{\langle act, \sigma \rangle} \mathcal{B}_2$, then $\mathcal{B}_1 \leftrightarrows \mathcal{B}_2$. Thus, the different target instances cannot be distinguished in our abstract domain. This notion of steps is similar to what the Chase does in existential rule reasoning [13]. Due to the closure of the domain under homomorphisms, it also resembles the standard chase and the core chase to certain extents [12]. Sets of concrete instances still proceed as originally defined in Sect. 2. To still guarantee a resemblance between the action labels in our abstract domain and the labels used for concrete instances (where no nulls are involved), we introduce a notion of *compatibility* of action labels.

Definition 9. *Action label $\langle act_1, \sigma_1 \rangle$ is* compatible to *action label $\langle act_2, \sigma_2 \rangle$, denoted by $\langle act_1, \sigma_1 \rangle \trianglelefteq \langle act_2, \sigma_2 \rangle$, if $act_1 = act_2$ and $\sigma_1 \subseteq \sigma_2$.*

Note that we could have reduced the action labeling to include only the guard matches for concrete instances already. However, this simplification does not make the branching finite. Even worse, the resulting LTS would become nondeterministic and looses image-finiteness at the same time.

As before, the abstraction mechanisms we study are based on greatest lower bounds and least upper bounds of the abstract domain $(\mathbb{I}^{\mathbf{N}}, \to)$. Next, we study the intersection abstraction of $C \in \mathbf{2}^{\mathbb{U}}$ with $\alpha_4(C)$ in (4). Generalizing from $(\mathbb{I}, \supseteq)$ we get $(\mathbb{I}^{\mathbf{N}}, \leftarrow)$ as the less databases C contains, the bigger the abstract instance becomes (cf. Def. 6 item 1). Conversely, $\gamma_4(\mathcal{I})$ in (4) for abstract instance $\mathcal{I} \in \mathbb{I}^{\mathbf{N}}$.

$$\alpha_4(C) := \bigsqcap C \qquad\qquad \gamma_4(\mathcal{I}) := \{\mathcal{D} \in \mathbb{U} \mid \mathcal{I} \to \mathcal{D}\} \tag{4}$$

Proposition 2. (α_4, γ_4) *is a Galois connection.*

Using labeled nulls, abstract DMSs using CGs become precise abstractions of their concrete counterparts.

Example 6. We analyze the guard g_{af} (does there exist a friendship from x to someone) from example 1 on $C = \{\{P(\mathrm{A}), P(\mathrm{B}), F(\mathrm{A}, \mathrm{B})\}, \{P(\mathrm{A}), P(\mathrm{C}), F(\mathrm{A}, \mathrm{C})\}\}$ and $\mathcal{I} = \bigsqcap C = \{P(\mathrm{A}), F(\mathrm{A}, \mathbf{n}_1)\}$. In contrast to $\mathcal{I}' = \bigcap C = \{P(\mathrm{A})\}$, we have a friendship with nulls in $\mathcal{I}$. Now we get homomorphisms $h_\mathcal{I} : g_{af} \to \mathcal{I}$ and $h_\mathcal{D} : g_{af} \to \mathcal{D}$ for each $\mathcal{D} \in C$.

Theorem 4. (α_4, γ_4) *is an abstract interpretation for CGs.*

Generalizing the Galois connection (α_1, γ_1) to $\mathbb{I}^\mathbf{N}$ yields (α_5, γ_5) with $\alpha_5 = \alpha_1$ and $\gamma_5(\mathcal{A}) := \{\mathcal{D} \in \mathbb{U}^\mathbf{N} \mid \mathcal{D} \to \mathcal{A}\}$. As for all databases $\mathcal{D}$ without labeled nulls, the existence of a homomorphism from $\mathcal{D}$ to $\mathcal{A}$ holds if, and only if, $\mathcal{D} \subseteq \mathcal{A}$, the new domain generalizes the original result (i.e., Theorem 1) slightly, but without further impact. After all, labeled nulls are proxies for the existence of constants, whereas CNA guards account for the absence of atoms.

Theorem 5. *Galois connection* (α_5, γ_5) *with* $\alpha_5(C) := \bigsqcup C$ *and* $\gamma_5(\mathcal{I}) := \{\mathcal{D} \in \mathbb{U} \mid \mathcal{D} \to \mathcal{I}\}$ *is an abstract interpretation for CNAs.*

4.3 Combining Unions and Intersections

Although the former abstractions already capture existentially quantified variables (i.e., projections), they do not jointly support projections as well as negation. A corresponding abstraction capturing both is $\alpha_6 : \mathbf{2}^\mathbb{U} \to \mathbb{I}^\mathbf{N} \times \mathbb{I}^\mathbf{N}$ with respective concretization $\gamma_6 : \mathbb{I}^\mathbf{N} \times \mathbb{I}^\mathbf{N} \to \mathbf{2}^\mathbb{U}$ as defined in (5). The abstract domain is $(\mathbb{I}^\mathbf{N} \times \mathbb{I}^\mathbf{N}, \preceq)$. $\mathcal{I}_1 \preceq \mathcal{I}_2$ is defined as $(\mathcal{I}_1^\sqcap, \mathcal{I}_1^\sqcup) \preceq (\mathcal{I}_2^\sqcap, \mathcal{I}_2^\sqcup)$ if and only if $\mathcal{I}_1^\sqcap \leftarrow \mathcal{I}_2^\sqcap$ and $\mathcal{I}_1^\sqcup \to \mathcal{I}_2^\sqcup$. The lattice $(\mathbb{I}^\mathbf{N} \times \mathbb{I}^\mathbf{N}, \preceq)$ is a combination of the two lattices $(\mathbb{I}^\mathbf{N}, \leftarrow)$ and $(\mathbb{I}^\mathbf{N}, \to)$.

$$\alpha_6(C) := (\bigsqcap C, \bigsqcup C) \qquad \gamma_6(\mathcal{I}) := \{\mathcal{I}^\sqcap \to \mathcal{D} \to \mathcal{I}^\sqcup\} \tag{5}$$

Proposition 3. (α_6, γ_6) *is a Galois connection.*

A substitution σ holds for a NCG g and an abstract state $\mathcal{I} = (\mathcal{I}^\sqcap, \mathcal{I}^\sqcup)$ if the following holds: $\sigma \in g(\mathcal{I})$ if $\sigma \in g^+(\mathcal{I}^\sqcap)$ and $\sigma \in g^-(\mathcal{I}^\sqcup)$.

Theorem 6. (α_6, γ_6) *is an abstract interpretation for NCG.*

Example 7. With Galois connection (α_6, γ_6) the guard $g_{end} := \exists x, y, z.F(x, y) \wedge F(y, z) \wedge F(z, x) \wedge \neg P(x, y) \wedge \neg P(y, z) \wedge \neg P(z, x)$ from action act_{end} (example 3) holds in the abstract and concrete domain.

5 Related Work

Reasoning about Database-Manipulating Processes. Most recent works consider formal process languages for manipulating relational database in the context of business process modeling [6].

Data manipulating systems (DMS) as considered in this paper are based on Abdullah et al. [1]. The authors use the formalism to study boundaries of decidability of (generally undecidable) reachability of state predicates in DMS processes. Their approach employs a formal semantics of DMS processes based on Petri nets and counter machines in combination with multiset-based abstraction of databases. Thereupon, Abdullah et al. impose bounds on database schemas as well as query evaluation to obtain decidable fragments. Calvanese et al. [7] also consider a DMS-like language for which they define an *LTS-based operational semantics* to support CTL model-checking of such systems. Similar to Abdullah et al., bounds are imposed on the generally infinite state space to enable an effective, yet incomplete model-checking procedure.

Cangialosi et al. [8,11] consider a DMS-like formalism called artifact-centric (service) language to verify process properties expressed in the μ-calculus. To obtain an effective verification procedure, the authors employ, in accordance to our framework, homomorphism equivalence as abstraction and restrict the process language to conjunctive queries, respectively. Bagheri et al. [4] extend the work of Cangialosi et al. by supporting negation within first-order queries serving as preconditions (guards) of transitions. As a consequence, processes must be restricted to be weakly acyclic in order to ensure a finite solution.

Other works use *Petri nets* with data (colored Petri nets) as a DMS-like formalism. Montali et al. [22] propose DB-nets to integrate data- and process-related aspects of business processes. In [21], Montali et al. adopt soundness checks (including reachability) known from workflow nets to DB-nets, where a finite solution is ensured by employing different notions of boundedness. This work work has recently been extended by Ghilardi et al. [14,15] to support conjunctive queries with atomic negation and existential quantifiers.

To summarize, most works impose bounds on the state space and/or restrictions of guard/query languages to ensure effective reasoning about semantic properties of DMS-like processes. However, to the best of our knowledge, none of these works provide a comprehensive decomposition hierarchy of guard/query expressions together with a precise characterization of corresponding semantic-preserving abstractions.

Abstraction Techniques for Databases. Halder et al. [16,17] apply principles of *abstract interpretation* in a more practical setting to define fine-grained abstractions for SQL query expressions. For approximating query result sets, query- and database-specific lattice-based abstractions are applied to value ranges of attribute constraints in selection conditions. In other works, abstract interpretation is mostly used to formalize the interface between database languages and programming languages. Baily et al. [5] apply abstract interpretation for termination analysis for a functional programming language performing database manipulations. Similar attempts are proposed by Amato et al. [3] and Toman et al. [25]

to reason about the interplay between imperative programming and database manipulating operations. However, using abstract interpretation to characterize an implementation-independent hierarchy of database abstractions as proposed in this paper has not yet been considered.

Besides abstract interpretation, *symbolic execution* techniques are also frequently considered to effectively cope with large/infinite state spaces of database systems. In these approaches, sets of databases instances are symbolically represented using logical constraints, where most recent works employ this approach for test-data generation from/for databases [23,20,18,19]. In contrast, elaborating a hierarchy of *symbolic* abstractions using different fragments of propositional logics similar to our approach, has not been investigated so far.

6 Conclusion

We proposed a hierarchy of abstract domains for representing (possibly infinite) sets of databases instances in a final way based on the principles of abstract interpretation. The resulting hierarchy is semantic-preserving up-to bisimilarity and is shaped by different fragments of first-order logics serving as guard language of database-manipulating processes. As a future work, our framework can be instantiated in different ways to facilitate DMS model-checking (e.g., considering corresponding fragments of the modal μ-calculus as specification language). To this end, a purely abstract step semantics is to be defined which allows us to explore the abstract LTS (e.g., starting from all possible initial database instances). We further plan to enrich DMS by a formal process language like Petri nets and CCS to investigate effects as induced by constructs like guarded choice and concurrent actions.

Acknowledgements. Stephan Mennicke has been partly supported by Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) in project number 389792660 (TRR 248, Center for Perspicuous Computing), by the Bundesministerium für Bildung und Forschung (BMBF, Federal Ministry of Education and Research) under project 13GW0552B (KIMEDS), in the Center for Scalable Data Analytics and Artificial Intelligence (ScaDS.AI), and by BMBF and DAAD (German Academic Exchange Service) in project 57616814 (SECAI, School of Embedded and Composite AI).

References

1. Abdulla, P.A., Aiswarya, C., Atig, M.F., Montali, M., Rezine, O.: Complexity of reachability for data-aware dynamic systems. In: ACSD. pp. 11–20. IEEE (2018)
2. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases, vol. 8. Addison-Wesley (1995)
3. Amato, G., Giannotti, F., Mainetto, G.: Data sharing analysis for a database programming language via abstract interpretation. In: VLDB. pp. 405–415 (1993)
4. Bagheri Hariri, B., Calvanese, D., De Giacomo, G., De Masellis, R., Felli, P.: Foundations of relational artifacts verification. In: BPM. pp. 379–395. Springer (2011)

5. Bailey, J., Poulovassilis, A.: Abstract interpretation for termination analysis in functional active databases. J. IIS **12**, 243–273 (1999)
6. Calvanese, D., De Giacomo, G., Montali, M.: Foundations of data-aware process analysis: A database theory perspective. In: PODS. pp. 1–12. ACM (2013)
7. Calvanese, D., Montali, M., Patrizi, F., Rivkin, A.: Implementing data-centric dynamic systems over a relational dbms. In: FDM. vol. 1378, pp. 209–212. CEUR-WS (2015)
8. Cangialosi, P., De Giacomo, G., De Masellis, R., Rosati, R.: Conjunctive artifact-centric services. In: ICSOC. pp. 318–333. Springer (2010)
9. Cousot, P., Cousot, R.: Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: POPL. p. 238–252. ACM (1977)
10. Dams, D., Gerth, R., Grumberg, O.: Abstract interpretation of reactive systems. TOPLAS **19**(2), 253–291 (1997)
11. De Giacomo, G., De Masellis, R., Rosati, R.: Verification of conjunctive artifact-centric services. Intl. J. of CIS **21**(02), 111–139 (2012)
12. Deutsch, A., Nash, A., Remmel, J.: The chase revisited. In: PODS. pp. 149–158. ACM (2008)
13. Fagin, R., Kolaitis, P.G., Miller, R.J., Popa, L.: Data exchange: Semantics and query answering. TCS **336**(1), 89–124 (2005)
14. Ghilardi, S., Gianola, A., Montali, M., Rivkin, A.: Petri nets with parameterised data: Modelling and verification. In: BPM. pp. 55–74. Springer (2020)
15. Ghilardi, S., Gianola, A., Montali, M., Rivkin, A.: Petri net-based object-centric processes with read-only data. IS **107**, 102011 (2022)
16. Halder, R., Cortesi, A.: Abstract interpretation for sound approximation of database query languages. In: INFOS. pp. 1–10. IEEE (2010)
17. Halder, R., Cortesi, A.: Abstract interpretation of database query languages. CLSS **38**(2), 123–157 (2012)
18. Li, C., Csallner, C.: Dynamic symbolic database application testing. In: DBTest (2010)
19. Lo, E., Cheng, N., Hon, W.K.: Generating databases for query workloads. VLDB Endowment **3**(1-2), 848–859 (2010)
20. Marcozzi, M., Vanhoof, W., Hainaut, J.L.: A relational symbolic execution algorithm for constraint-based testing of database programs. In: SCAM. pp. 179–188. IEEE (2013)
21. Montali, M., Rivkin, A.: Model checking petri nets with names using data-centric dynamic systems. FAOC **28**(4), 615–641 (2016)
22. Montali, M., Rivkin, A.: Db-nets: On the marriage of colored petri nets and relational databases. TOPNOC pp. 91–118 (2017)
23. Pan, K., Wu, X., Xie, T.: Database state generation via dynamic symbolic execution for coverage criteria. In: DBtest. pp. 1–6 (2011)
24. Schüler, T., Mennicke, S., Lochau, M.: Abstract Domains for Database Manipulating Processes. CoRR **abs/2308.03466** (2023)
25. Toman, D.: Constraint databases and program analysis using abstract interpretation. In: CDB. pp. 246–262. Springer (1997)