Decidable Verification of Agent-Based Data-aware Systems

Francesco Belardinelli[1,2] and Vadim Malvone[2]

[1] Imperial College London, United Kingdom
[2] Laboratoire IBISC, Universite d'Evry, France

Abstract. In recent years the area of knowledge representation and reasoning (KR&R) has witnessed a growing interest in the modelling and analysis of data-driven/data-centric systems. These are systems in which the two tenets of data and processes are given equal importance, differently from traditional approaches whereby the data content is typically abstracted away in order to make the reasoning task easier. However, if data-aware systems (DaS) are to be deployed in concrete KR&R scenarios, it is key to develop tailored verification techniques, suitable to account for both data and processes. In this contribution we consider for the first time to our knowledge the parameterised verification of DaS. In particular, we prove that – under specific assumptions – this problem is decidable by computing a suitable cut-off value. We illustrate the proposed approach with a use case from the literature on businness process modelling.

1 Introduction

The ever increasing reliance of AI technologies on data acquisition, managements, and processing is having a profound impact on the nature and mission of artificial intelligence itself [28]. In recent years the area of knowledge representation and reasoning (KR&R) has witnessed a growing interest in the modelling and analysis of data-driven/data-centric/data-intensive systems [16,15,3]. This paradigm shift towards data-aware systems (DaS) has initiated in the area of business process modelling (BPM), in response to traditional approaches to service-oriented computing that typically abstract the data content away to reduce the complexity of the system description [30]. However, this data content is often essential to drive a business process. Hence, according to the data-aware perspective on BPM, the *data content* and the *processes* operating on it are seen as two equally relevant tenets in modelling systems [20,11]. This data-aware approach has proved fruitful also in applications to areas in KR&R, including commitments in negotiation [27], planning [9], and service-oriented computing [14], where processes are often thought of as *agents*, endowed with their own goals, plans to achieve them, as well as information about the external environment [29].

Yet, if agent-based DaS are to be deployed in concrete KR&R scenarios, it is key to develop verification techniques, suitable to account for the two tenets of data and processes. Then, a critical issue in tackling this task lies in the infinite state space generated by the possibly infinite data content of DaS. Recently, several contributions have addressed this problem [3,7,26,10], also leading to the development of open-source toolkits for DaS verification [25,19]. Nonetheless, we identify a conceptual difficulty

with most of the current approaches in the literature: data-aware systems are typically assumed to contain an actual infinity of data and to be able to reason about such an actual infinity. For instance, in [7,10] an infinite quantification domain is part of the system's description. But real-life scenarios actually deal only with a finite, possibly *unbounded*, quantity of data. Hence, the soundness and applicability of those theoretical results to concrete DaS scenarios cannot be taken for granted.

To provide an answer to the difficulties pertaining to reasoning about an actual infinite data domain, in Section 2 we introduce parameterised agent-based DaS (or P-AbDaS) as abstract systems, which are to be coupled with a (finite) data domain, in order to generate a concrete agent-based DaS (or C-AbDaS). Hence, differently from [7,10], the same P-AbDaS can be instantiated in possibly infinitely-many C-AbDaS, but all of them are finite. Further, to specify the behaviour of P-AbDaS we need both temporal operators to describe the system's evolution, and first-order features, including quantifiers and relation symbols, to account for data. Hence, in Section 3 we consider a first-order extension of the computation-tree logic CTL as the specification language for P-AbDaS, and then define the *parameterised* model checking problem for this setting, which we show to be undecidable in general. Then, in Section 4 we introduce techniques based on isomorphisms and finite interpretation that allow – under specific assumptions – for the existence of a *cut-off*, that is, a bound on the size of the quantification domain above which the truth value of formulas in first-order CTL does not change. The existence and value of the cut-off allow for a complete model checking procedure that checks the specification on increasingly larger domains, up to the cut-off value. We illustrate the formal machinery with a procurement scenario from the literature on BPM [21]. Finally, we conclude in Section 5 by discussing related work and pointing to future directions of research.

2 Agent-based Data-aware Systems

In this section we introduce parameterised agent-based data-aware systems (P-AbDaS) and define the corresponding model checking problem w.r.t. a first-order version of the temporal logic CTL. We first present the basic terminology on databases that is used throughout the paper [1].

Definition 1 (Database schema and instance). *A* database schema *is a finite set* $\mathcal{D} = \{P_1/q_1, \ldots, P_n/q_n\}$ *of* relation symbols P *with arity* $q \in \mathbb{N}$.

Given a countable interpretation domain Y*, a* $\mathcal{D}$-instance *over* Y *is a mapping* D *associating each relation symbol* P *to a* finite *q-ary relation on* Y*, i.e.,* $D(P) \underset{fin}{\subset} Y^q$.

By Def. 1 a database instance can be thought of as a finite relational structure, in line with relational models of databases [1]. We denote the set of all $\mathcal{D}$-instances on domain Y as $\mathcal{D}(Y)$. The *active domain* $adom(D)$ of a $\mathcal{D}$-instance D is the *finite* set of all elements $u \in Y$ occurring in some predicate interpretation $D(P)$, that is, $adom(D) = \bigcup_{P \in \mathcal{D}} \{u \in Y \mid \langle u_1, \ldots, u, \ldots, u_q \rangle \in D(P)\}$. Hereafter, we assume w.l.o.g. that the active domain also includes a finite set $C \subseteq Y$ of constants, i.e., $C \subseteq adom(D)$. To describe the temporal evolution of agent-based data-aware systems, we introduce the *primed version* of a database schema $\mathcal{D}$ as the schema $\mathcal{D}' = \{P'_1/q_1, \ldots, P'_n/q_n\}$.

Then, the *disjoint union* $D \oplus D'$ of $\mathcal{D}$-instances D and D' is the $(\mathcal{D} \cup \mathcal{D}')$-instance such that *(i)* $(D \oplus D')(P_i) = D(P_i)$, and *(ii)* $(D \oplus D')(P'_i) = D'(P_i)$, where $\mathcal{D}'$ is the primed version of $\mathcal{D}$. Intuitively, D and D' represent the current and next state of the system respectively, represented as database instances.

To specify properties of databases, we now recall the syntax of first-order logic with equality and no function symbols. Let V be a countable set of individual variables and let a *term* be any element $t \in T = V \cup C$.

Definition 2 (FO-formulas). *Given a database schema $\mathcal{D}$, the formulas φ of the first-order language $\mathcal{L}_\mathcal{D}$ are defined by the following BNF:*

$$\varphi ::= P(t_1, \ldots, t_q) \mid t = t' \mid \neg\varphi \mid \varphi \rightarrow \varphi \mid \forall x \varphi$$

where $P \in \mathcal{D}$, $t_1, \ldots, t_q$ is a q-tuple of terms, and t, t' are terms.

We define the free and bound variables in a formula φ as standard, and write $\varphi(\boldsymbol{x})$ to denote that the free variables of φ are among $x_1, \ldots, x_n$.

To interpret first-order formulas on database instances, we introduce *assignments* as functions $\sigma : T \rightarrow Y$ from terms to elements in Y. We denote by σ^x_u the assignment such that *(i)* $\sigma^x_u(x) = u$; and *(ii)* $\sigma^x_u(x') = \sigma(x')$ for every $x' \neq x$. Also, we assume a Herbrandian interpretation of constants, that is, $\sigma(c) = c$ for all $c \in C$.

Definition 3 (Satisfaction of FO-formulas). *Given a $\mathcal{D}$-instance D, an assignment σ, and an FO-formula $\varphi \in \mathcal{L}_\mathcal{D}$, we inductively define whether D satisfies φ under σ, or $(D, \sigma) \models \varphi$, as follows:*

$(D, \sigma) \models P(t_1, \ldots, t_q)$	*iff*	$\langle \sigma(t_1), \ldots, \sigma(t_q) \rangle \in D(P)$
$(D, \sigma) \models t = t'$	*iff*	$\sigma(t) = \sigma(t')$
$(D, \sigma) \models \neg\varphi$	*iff*	$(D, \sigma) \not\models \varphi$
$(D, \sigma) \models \varphi \rightarrow \varphi'$	*iff*	$(D, \sigma) \not\models \varphi$ *or* $(D, \sigma) \models \varphi'$
$(D, \sigma) \models \forall x \varphi$	*iff*	*for all* $u \in$ *adom*(D), $(D, \sigma^x_u) \models \varphi$

A formula φ is true *in D, or $D \models \varphi$, iff $(D, \sigma) \models \varphi$ for all assignments σ.*

Notice that we adopt an *active domain* semantics, where quantifiers range over the active domain *adom*(D) of D. This is a standard assumption in database theory [1]. Hereafter, we often write $(D, \boldsymbol{u}) \models \varphi$ whenever $\boldsymbol{x}$ are all the free variables in φ and $\sigma(\boldsymbol{x}) = \boldsymbol{u}$. In particular, the satisfaction of a formula only depends on its free variables.

We now introduce a notion of agent whose local information state is represented as a relational database. In particular, inspired by the literature in KR&R and BPM on the specification of agent actions in terms of pre- and post-conditions [2,3,21], we introduce the notion of *action type*.

Definition 4 (Action Type). *An* action type *is an expression $\alpha(\boldsymbol{x}) ::= g(\boldsymbol{x}) \rightsquigarrow ef(\boldsymbol{x})$, where:*

- *guard g is an FO-formula with free variables $\boldsymbol{x}$;*
- *effect ef is an expression built according to the BNF:*

$$ef ::= add(P, \boldsymbol{x}) \mid del(P, \boldsymbol{x}) \mid ef; ef \mid ef \cup ef$$

where, intuitively, $add(P, \boldsymbol{x})$ is the insertion of tuple $\boldsymbol{x}$ in relation P, $del(P, \boldsymbol{x})$ is the deletion of $\boldsymbol{x}$ from P, $ef\,; ef$ is the sequential composition, and $ef \cup ef$ is the non-deterministic choice.

We now introduce a set Ag of agents, operating on databases, each of them defined as follows:

Definition 5 (Agent). *An* agent *is a tuple $i = \langle \mathcal{D}_i, Act_i \rangle$, where*

- *$\mathcal{D}_i$ is the* local database schema*;*
- *Act_i is the finite set of* action types *$\alpha(\boldsymbol{x})$, whose guards and effects are built over $\mathcal{D}_i$.*

Intuitively, by Def. 5 we assume that at each moment agent i is in some local state $D \in \mathcal{D}_i(Y)$ that represents all the information she has about the global state of the system. In this respect we follow the typical approach to agent-based systems [17,31], but here we require that this information is structured as a database. Further, each agent has her own database schema $\mathcal{D}_i$, but the same relation symbol might appear in several schemas.

As we are interested in the interactions of agents among themselves and with the external environment, we introduce their synchronous composition.

Definition 6 (Parameterised AbDaS). *A* parameterised agent-based data-aware system *(or P-AbDaS) is a finite set Ag of agents defined as in Def.5.*

To endow a P-AbDaS with a data content, thus obtaining a concrete Ab-DaS, we consider an infinite, countable interpretation domain $\mathcal{Y}$, which intuitively represents these data.

Definition 7 (Concrete AbDaS). *A* concrete agent-based data-aware system *(or C-AbDaS) is a tuple $\mathcal{P} = (Ag, Y)$, where* (i) *Ag is a P-AbDaS; and* (ii) *$Y \supseteq C$ is a* finite *subset of $\mathcal{Y}$.*

Notice that, differently from [7,10], we do not assume an actual infinity of elements in our models: each C-AbDaS only contains a finite set Y of elements. However, in general we can obtain infinitely many C-AbDaS based on the same P-AbDaS, build on different domains $Y \underset{fin}{\subset} \mathcal{Y}$.

We now introduce some technical notions that will be used in the rest of the paper. Given a C-AbDaS $\mathcal{P} = (Ag, Y)$, the (global) *states* of $\mathcal{P}$ are tuples $s \in S = \prod_{i \in Ag} \mathcal{D}_i(Y)$, whereas *joint actions* $\alpha(\boldsymbol{u}) \in ACT(Y) = \prod_{i \in Ag} Act_i(Y)$ take values $\boldsymbol{u}$ from domain Y. Observe that every global state $s = \langle D_0, \ldots, D_n \rangle \in S$ can be thought of as a database instance on the *global* database schema $\mathcal{D} = \bigcup_{i \in Ag} \mathcal{D}_i$ such that $s(P) = \bigcup_{i \in Ag} \mathcal{D}_i(P)$, for every $P \in \mathcal{D}$. Then, we set s_i as the restriction of s to the relation symbols in $\mathcal{D}_i$. That is, we assume that each agent has a truthful, yet partial, view of the global database $\mathcal{D}$, since in general $\mathcal{D}_i$ is a subset of $\mathcal{D}$.

Further, the *transition relation* $\tau : S \times ACT(Y) \mapsto 2^S$ is defined such that $t = \langle D'_0, \ldots, D'_n \rangle \in \tau(s, \alpha(\boldsymbol{u}))$ iff for every $i \in Ag$, $(s_i, \boldsymbol{u}) \models g_i$, i.e., all guards are satisfied and the corresponding joint action is enabled, and applying the effects $ef_i(\boldsymbol{u})$.

Specifically, if $ef_i = add(P, \boldsymbol{x})$ (resp. $del(P, \boldsymbol{x})$), then D'_i is obtained from D_i by performing the corresponding insertion (resp. deletion) in P with values $\boldsymbol{u}$. If $ef_i = ef'_i; ef''_i$, then t_i is obtained from s_i by applying first the effects in ef'_i, and then ef''_i. Similarly for $ef_i = ef'_i \cup ef''_i$.

Finally, we introduce the *successor relation* $\rightarrow$ on global states such that $s \rightarrow t$ if there exists $\alpha(\boldsymbol{u}) \in ACT(Y)$ such that $s \xrightarrow{\alpha(\boldsymbol{u})} t$, i.e., $t \in \tau(s, \alpha(\boldsymbol{u}))$. A *run* r from state s is an infinite sequence $s^0 \rightarrow s^1 \rightarrow \ldots$, with $s^0 = s$. For $n \in \mathbb{N}$, we define $r(n) = s^n$. Hereafter we assume that the relation $\rightarrow$ is serial. This can be ensured by using skip actions. Notice that, in what follows we restrict the set of global states as the set of reachable states only. The disjoint union $\oplus$ is extended to global states in a pointwise manner: for $s = \langle D_0, \ldots, D_n \rangle$ and $s' = \langle D'_0, \ldots, D'_n \rangle$, we define $s \oplus s'$ as $\langle D_0 \oplus D'_0, \ldots, D_n \oplus D'_n \rangle$.

Example 1. To illustrate the formal machinery introduced thus far, we present a business process inspired by a concrete IBM customer use case [21]. The order-to-cash business process specifies the interactions of three agents in an e-commerce situation relating to the purchase and delivery of a product: a manufacturer m, a customer c, and a supplier s. The process begins when c prepares and submits to m a *purchase order* (PO), i.e., a list of products c requires (action $createPO()$). Upon receiving a PO, m prepares a *material order* (MO), i.e., a list of components needed to assemble the requested products (action $createMO()$). Then, m forwards to s the relevant material order. Upon receiving an MO, s can either accept or reject it (actions $acceptMO()$ and $rejectMO()$). In the former case she proceeds to deliver the requested components to m (action $shipMO()$). In the latter, she notifies m of her rejection. If an MO is rejected, m deletes it and then prepares and submits a new MO (action $deleteMO()$). Upon delivery of the components (action $receiveMO()$), m assembles the product and, provided the order has been paid for (action $payPO()$), delivers it to c (action $shipPO()$).

We can encode the order-to-cash business process as a P-AbDaS, where the data model is represented by means of database schemas, whose evolution is determined by an appropriate set of actions types. Formally the three agents can be defined as follows:

- $A_c = \langle \mathcal{D}_c, Act_c \rangle$, where
  - $\mathcal{D}_c\{Products(prod\_code, budget), PO(id, prod\_code, offer, status)\}$;
  - $Act_c = \{createPO(id, code), payPO(id), deletePO(id)\}$;
- $A_m = \langle \mathcal{D}_m, Act_m \rangle$, where
  - $\mathcal{D}_m$=$\{PO(id, prod\_code, offer, status)$,$MO(id, prod\_code, price, status)\}$;
  - Act_m=$\{createMO(id, price)$,$receiveMO(id)$,$deleteMO(id)$,$shipPO(id)\}$;
- $A_s = \langle \mathcal{D}_s, Act_s \rangle$, where
  - $\mathcal{D}_s = \{Materials(mat\_code, cost), MO(id, prod\_code, price, status)\}$;
  - $Act_s = \{acceptMO(id), rejectMO(id), shipMO(id)\}$.

In Table 1 we provide the detailed action types for all agents in the use case. As an example, according to action type $createPO()$ (item (1.a)), the customer can create a purchase order with a designed id only if there exists a product with the same id. Further, by using $createMO()$ the manufacturer can create a material order with a

The actions of customer c:

1. $createPO(id, code) ::= Products(code, x) \wedge \neg\exists z PO(id, code, z, \mathsf{submitted}) \rightsquigarrow add(PO(id, code, x, \mathsf{submitted}))$
2. $payPO(id) ::= (PO(id, x, y, \mathsf{prepared}) \wedge PO(id, x, y', \mathsf{submitted}) \wedge y = y') \rightsquigarrow del(PO(id, x, y, \mathsf{submitted})); add(PO(id, x, y, \mathsf{paid}))$
3. $deletePO(id) ::= PO(id, x, y, \mathsf{shipped}) \rightsquigarrow del(PO(id, x, y, \mathsf{paid}))$

The actions of manufacturer m:

1. $createMO(id, price) ::= (PO(id, x, offer, \mathsf{submitted}) \wedge \neg\exists z MO(id, z, price, \mathsf{preparation})) \rightsquigarrow add(MO(id, x, price, \mathsf{preparation}))$
2. $receiveMO(id) ::= MO(id, x, y, \mathsf{shipped}) \rightsquigarrow del(MO(id, x, y, \mathsf{preparation})); add(MO(id, x, y, \mathsf{received})); add(PO(id, x, y, \mathsf{prepared}))$
3. $deleteMO(id) ::= MO(id, x, y, \mathsf{rejected}) \rightsquigarrow del(MO(id, x, y, \mathsf{preparation}))$
4. $shipPO(id) ::= PO(id, x, y, \mathsf{paid}) \rightsquigarrow del(PO(id, x, y, \mathsf{prepared})); add(PO(id, x, y, \mathsf{shipped}))$

The actions of supplier s:

1. $acceptMO(id) ::= MO(id, code, y, \mathsf{preparation}) \wedge \neg\exists z MO(id, code, z, \mathsf{accepted}) \wedge Materials(code, y) \rightsquigarrow add(MO(id, code, y, \mathsf{accepted}))$
2. $rejectMO(id) ::= MO(id, code, y, \mathsf{preparation}) \wedge \neg\exists z MO(id, code, z, \mathsf{rejected}) \wedge \neg Materials(code, y) \rightsquigarrow add(MO(id, code, y, \mathsf{rejected}))$
3. $shipMO(id) ::= MO(id, x, y, \mathsf{accepted}) \rightsquigarrow del(MO(id, x, y, \mathsf{accepted})); add(MO(id, x, y, \mathsf{shipped}))$

Table 1: the list of actions in the order-to-cash scenario.

designed id if MO does not contain a tuple with same id in preparation status (item (2.a)).

3 The Verification of AbDaS

In this section we introduce the specification language for AbDaS and the corresponding model checking problem. We recall that we consider a set V of *individual variables* and a set C of *individual constants*. The terms $t_1, t_2, \ldots$ in T are either variables in V or constants in C.

Definition 8 (FO-CTL). *The FO-CTL formulas φ over a database schema $\mathcal{D}$ are defined as follows, where $P \in \mathcal{D}$:*

$$\varphi ::= P(t_1, \ldots, t_q) \mid \neg\varphi \mid \varphi \rightarrow \varphi \mid \forall x\varphi \mid AX\varphi \mid A\varphi U\varphi \mid E\varphi U\varphi$$

The language FO-CTL is a first-order extension of the propositional temporal logic CTL. The temporal formulas $AX\varphi$ and $A\varphi U\varphi'$ (resp. $E\varphi U\varphi'$) are read as "for all runs, at the next step φ" and "for all runs (resp. some run), φ until φ'". Given a formula φ, we denote the set of free and all variables as $fr(\varphi)$ and $var(\varphi)$ respectively, and introduce formulas $EX\varphi$, $AF\varphi$, $AG\varphi$, $EF\varphi$, and $EG\varphi$ as standard.

We now interpreted FO-CTL on concrete agent-based data-aware systems.

Definition 9 (Semantics of FO-CTL). *We define whether a C-AbDaS $\mathcal{P}$* satisfies *a formula φ in a state s according to assignment σ, or $(\mathcal{P}, s, \sigma) \models \varphi$, as follows:*

$(\mathcal{P}, s, \sigma) \models P(\boldsymbol{t})$ *iff* $\langle \sigma(t_1), \ldots, \sigma(t_q) \rangle \in s(P)$
$(\mathcal{P}, s, \sigma) \models t = t'$ *iff* $\sigma(t) = \sigma(t')$
$(\mathcal{P}, s, \sigma) \models \neg\varphi$ *iff* $(\mathcal{P}, s, \sigma) \not\models \varphi$
$(\mathcal{P}, s, \sigma) \models \varphi \rightarrow \varphi'$ *iff* $(\mathcal{P}, s, \sigma) \not\models \varphi$ *or* $(\mathcal{P}, s, \sigma) \models \varphi'$
$(\mathcal{P}, s, \sigma) \models \forall x\varphi$ *iff for all* $u \in adom(s)$, $(\mathcal{P}, s, \sigma^x_u) \models \varphi$
$(\mathcal{P}, s, \sigma) \models AX\varphi$ *iff for all* r, *if* $r(0) = s$ *then* $(\mathcal{P}, r(1), \sigma) \models \varphi$
$(\mathcal{P}, s, \sigma) \models A\varphi U\varphi'$ *iff for all* r, *if* $r(0) = s$ *then there is* $k \geq 0$ *s.t.* $(\mathcal{P}, r(k), \sigma) \models \varphi'$, *and for all* j, $0 \leq j < k$ *implies* $(\mathcal{P}, r(j), \sigma) \models \varphi$
$(\mathcal{P}, s, \sigma) \models E\varphi U\varphi'$ *iff for some* r, $r(0) = s$ *and there is* $k \geq 0$ *s.t.* $(\mathcal{P}, r(k), \sigma) \models \varphi'$, *and for all* j, $0 \leq j < k$ *implies* $(\mathcal{P}, r(j), \sigma) \models \varphi$

A formula φ is true *at state s, or $(\mathcal{P}, s) \models \varphi$, if $(\mathcal{P}, s, \sigma) \models \varphi$ for all assignments σ; φ is* true *in C-AbDaS $\mathcal{P}$, or $\mathcal{P} \models \varphi$, if $(\mathcal{P}, s) \models \varphi$ for every $s \in S$. Finally, $Ag \models \varphi$ iff $(Ag, Y) \models \varphi$ for all $Y \underset{fin}{\subset} \mathcal{Y}$.*

Again, in Def. 9 we adopt an active domain semantics, whereby quantifiers range over the active domain *adom*(s) of s.

Finally, we present the model checking problem for P-AbDaS with respect to the specification language FO-CTL.

Definition 10 (Parameterised Model Checking). *Given a P-AbDaS Ag, an infinite domain $\mathcal{Y}$, and an FO-CTL formula φ, determine whether $Ag \models \varphi$.*

Notice that the parameterised model checking problem requires in principle to check an infinite number of C-AbDaS built on the same P-AbDaS. Indeed, model checking P-AbDaS is undecidable in general: we remark without proof that P-AbDaS are expressive enough to encode Turing machines, and reachability of a halting state can then be expressed in FO-CTL similarly to [15,7]. Hence, it is of interest to investigate semantic restrictions on P-AbDaS that allow for a decidable model checking problem.

To this end, a key notion to decide parameterised model checking in general is the *cut-off*:

Definition 11 (Cut-off). *A natural number $n \in \mathbb{N}$ is a* cut-off *for P-AbDaS Ag and formula ϕ iff for all finite subsets $Y \supseteq C, Y' \supseteq C$ of $\mathcal{Y}$, if $|Y| = n$ and $|Y'| \geq |Y|$, then $(Ag, Y) \models \phi$ iff $(Ag, Y') \models \phi$*

Note that, in Def.11 we suppose $|Y'| \geq |Y|$ without considering that $|Y|$ is a subset of $|Y'|$. This is because we define the set of constants C to be in both $|Y|$ and $|Y'|$, and for this reason the intersection between $|Y'|$ and $|Y|$ cannot be empty.

The existence of the cut-off allows us to decide verification by checking all C-AbDaS up to size $|n|$, of which there exist finitely many instances. We devote the rest of the paper to finding sufficient condition for the existence of cut-offs.

We conclude this section by elaborating on Example 1.

Example 2. We can investigate properties of the order-to-cash business process by using specifications in FO-CTL. For instance, the following formula intuitively specifies that each material order MO has to match a corresponding purchase order PO:

$$AG\ \forall id, pc\ (\exists pr, s\ MO(id, pc, pr, s) \rightarrow \exists o, s' PO(id, pc, o, s'))$$

The next specification states that given a material order MO, it can be the case that eventually the corresponding PO will be shipped.

$$AG\ \forall id, pc\ (\exists pr, s\ MO(id, pc, pr, s) \rightarrow EF\ \exists o\ PO(id, pc, o, \mathsf{shipped}))$$

Hereafter we develop techniques to model check specifications in FO-CTL like the ones above.

4 Finding Cut-offs

In this section we introduce model-theoretic notions that will be used to tackle the parameterised model checking problem for P-AbDaS. In particular, we recall some notions in [7].

Definition 12 (Isomorphism). *Two database instances $D \in \mathcal{D}(Y')$, $D' \in \mathcal{D}(Y)$ are* isomorphic*, or $D \simeq D'$, iff there exists a bijection $\iota : adom(D) \mapsto adom(D')$ s.t.:*

(i) *ι is the identity on the constants in C;*
(ii) *for all $P \in \mathcal{D}$, $\boldsymbol{u} \in Y^q$, $\boldsymbol{u} \in D(P)$ iff $\iota(\boldsymbol{u}) \in D'(P)$.*

When the above is the case, we say that ι is a witness *for $D \simeq D'$. Moreover, two global states $s = \langle D_0, \ldots, D_n \rangle \in S$ and $s' = \langle D'_0, \ldots, D'_n \rangle \in S'$ are* isomorphic*, or $s \simeq s'$, iff there exists a bijection $\iota : adom(s) \mapsto adom(s')$ such that for every $j \in Ag$, ι is a witness for $D_j \simeq D'_j$.*

By Def. 12 isomorphisms preserve the interpretation of constants as well as of predicates up to renaming of terms. Obviously, isomorphisms are equivalence relations. Given a function $f : Y \mapsto Y'$ defined on $adom(s)$, $f(s)$ denotes the instance in $\mathcal{D}(Y')$ obtained from s by renaming each $u \in adom(s)$ as $f(u)$. If f is also injective (thus invertible) and the identity on C, then $f(s) \simeq s$.

While isomorphic states share the same relational structure, two isomorphic states do not necessarily satisfy the same FO-formulas as satisfaction depends also on the values assigned to free variables. To account for this, we introduce the following notion.

Definition 13 (Equivalent assignments). *Given states $s \in S$ and $s' \in S'$, and a set $V' \subseteq V$ of variables, assignments $\sigma : T \mapsto Y$ and $\sigma' : T \mapsto Y'$ are* equivalent for V' w.r.t. s and s' *iff there exists a bijection $\gamma : adom(s) \cup \sigma(V') \mapsto adom(s') \cup \sigma'(V')$ such that:*

(i) *$\gamma|_{adom(s)}$ is a witness for $s \simeq s'$;*

(ii) $\sigma'|_{V'} = \gamma; \sigma|_{V'}$, *where* ; *is function composition.*

By Def. 13 equivalent assignments preserve both the (in)equalities of the terms in s, s' up to renaming. Clearly, the existence of equivalent assignments implies that s, s' are isomorphic. We say that two assignments are *equivalent for an FO-CTL formula* φ, omitting states s and s' when clear from the context, if these are equivalent for the free variables $fr(\varphi)$ in φ.

We now state the following standard result in first-order (non-modal) logic, i.e., isomorphic states satisfy exactly the same FO-formulas, when interpreted with equivalent assignments [1].

Proposition 1. *Given isomorphic states $s \in S$ and $s' \in S'$, an FO-formula φ, and assignments σ and σ' equivalent for φ, we have that*

$$(s, \sigma) \models \varphi \textit{ iff } (s', \sigma') \models \varphi$$

An immediate consequence of Prop. 1 is that isomorphic states cannot be distinguished by FO-sentences. In the rest of the section we show how isomorphisms can actually be used to prove the preservation of the whole FO-CTL. Notice that this is in marked contrast with similar results in the literature [7,3], which need to assume some notion of (bi)simulation on the underlying transitions systems. Nothing similar is required here, we show that isomorphisms suffice. More specifically, in [7] the requirement of *uniformity* was put forward as a sufficient condition for bisimilar systems to satisfy the same formulas in FO-CTL. We now show that C-AbDaS satisfy uniformity unrestrictedly.

Lemma 1 (Uniformity). *All C-AbDaS $\mathcal{P}$, $\mathcal{P}'$ are* uniform*, that is, for every $s, t \in S$, $s' \in S'$, $t' \in \mathcal{D}(Y)$, if $t \in \tau(s, \alpha(\boldsymbol{u}))$ and $s \oplus t \simeq s' \oplus t'$ for some witness ι, then for every constant-preserving bijection ι' that extends ι to $\boldsymbol{u}$, we have that $t' \in \tau(s', \alpha(\iota'(\boldsymbol{u})))$.*

Proof. For illustration, we consider the case in which there is only one agent, i.e., $\alpha(\boldsymbol{u}) = g(\boldsymbol{u}) \rightsquigarrow ef(\boldsymbol{u})$. First of all, notice that if $s \oplus t \simeq s' \oplus t'$ then for every bijection ι' extending ι to $\boldsymbol{u}$, we have that $(s, \boldsymbol{u}) \models g(\boldsymbol{x})$ iff $(s, \iota'(\boldsymbol{u})) \models g(\boldsymbol{x})$ by Prop. 1. Hence, action $\alpha(\boldsymbol{u})$ is enabled in s iff $\alpha(\iota'(\boldsymbol{u}))$ is enabled in s'.

Now we prove by induction on the structure of $ef(\boldsymbol{u})$ that t' can be obtained by applying effects $ef(\iota'(\boldsymbol{u}))$ to s', and therefore $t' \in \tau(s', \alpha(\iota'(\boldsymbol{u})))$. For the base of induction, consider $ef(\boldsymbol{u}) = add(P, \boldsymbol{u})$. Then, t differs from s only for tuple $\boldsymbol{u}$ possibly added to the interpretation of P. Since $s \oplus t \simeq s' \oplus t'$, also t' differs from s' only for tuple $\iota'(\boldsymbol{u})$ added to the interpretation of P, and therefore $t' \in \tau(s', \alpha(\iota'(\boldsymbol{u})))$. As regards the base case for $ef(\boldsymbol{u}) = del(P, \boldsymbol{u})$, t differs from s only for tuple $\boldsymbol{u}$ possibly deleted from the interpretation of P. Since $s \oplus t \simeq s' \oplus t'$, again t' differs from s' only for tuple $\iota'(\boldsymbol{u})$ deleted from the interpretation of P, and therefore $t' \in \tau(s', \alpha(\iota'(\boldsymbol{u})))$.

As for the inductive case for $ef(\boldsymbol{u}) = ef_1(\boldsymbol{u}_1) \cup ef_2(\boldsymbol{u}_2)$, then t is obtained from s by applying either the effects in $ef_1(\boldsymbol{u}_1)$ or in $ef_2(\boldsymbol{u}_2)$. Then, by induction hypothesis, t' can be obtained from s' by applying either the effects in $ef_1(\iota'(\boldsymbol{u}_1))$ or in $ef_2(\iota'(\boldsymbol{u}_2))$, which is tantamount to $ef(\iota'(\boldsymbol{u}))$. Finally, for $ef(\boldsymbol{u}) = ef_1(\boldsymbol{u}_1); ef_2(\boldsymbol{u}_2)$, t is obtained from s by applying first the effects in $ef_1(\boldsymbol{u}_1)$ and then $ef_2(\boldsymbol{u}_2)$. Then, by induction

hypothesis, t' can be obtained from s' by applying first the effects in $ef_1(\iota'(\boldsymbol{u}_1))$ and then $ef_2(\iota'(\boldsymbol{u}_2))$, which is tantamount to $ef(\iota'(\boldsymbol{u}))$.

Intuitively, the notion of uniformity in Lemma 1 captures the idea that actions take into account and operate only on the relational structure of states, irrespective of the actual data they contain. Because of this, uniformity has been compared to the notion of *genericity* in database theory, whereby in specific cases the answer to a query depends only on the structure of the database [1]. Actually, the result in Lemma 1 is stronger that the notion of uniformity in [7], which is restricted to states belonging to the same system. We are able to prove a stronger result, as we consider C-AbDaS built on the same P-AbDaS and therefore sharing the same actions, which is not the case in [7].

We now demonstrate some auxiliary lemmas that will be used in proving the main preservation result (Theorem 2). The first two guarantee that under appropriate conditions on the cardinality of the interpretation domains, equivalent assignments are preserved by the isomorphism relation. Hereafter we set $N_{Ag} = \sum_{i \in Ag} \max_{\alpha(\boldsymbol{x}) \in Act_i} \{|\boldsymbol{x}|\}$, i.e., N_{Ag} is the sum of the maximum number of parameters contained in the action types of each agent in Ag; whereas $\mathcal{P} = (Ag, Y)$ and $\mathcal{P}' = (Ag, Y')$ are C-AbDaS defined on the same P-AbDaS Ag.

Lemma 2. *Consider C-AbDaS $\mathcal{P}$ and $\mathcal{P}'$ defined on the same P-AbDaS Ag, isomorphic states $s \in S$ and $s' \in S'$, an FO-CTL formula φ, and assignments σ and σ' equivalent for φ w.r.t. s and s'. For every $t \in S$ such that $s \to t$, if $|Y'| \geq |adom(s) \cup \sigma(fr(\varphi))| + N_{Ag}$, then there exists $t' \in S'$ such that $s' \to t'$, $t \simeq t'$, and σ and σ' are equivalent for φ w.r.t. t and t'.*

Proof. First of all, let γ be a bijection witnessing that σ and σ' are equivalent for φ w.r.t. s and s', and suppose that $t \in \tau(s, \boldsymbol{\alpha}(\boldsymbol{u}))$ for some joint action $\boldsymbol{\alpha}(\boldsymbol{u})$. Now define $Dom(j) \doteq adom(s) \cup \sigma(fr(\varphi)) \cup \boldsymbol{u}$, and partition it into:

- $Dom(\gamma) \doteq adom(s) \cup \sigma(fr(\varphi))$;
- $Dom(\iota') \doteq \boldsymbol{u} \setminus Dom(\gamma)$.

Let $\iota' : Dom(\iota') \mapsto Y' \setminus Im(\gamma)$ be an invertible total function. Observe that $|Im(\gamma)| = |adom(s') \cup \sigma'(fr(\varphi))| = |adom(s) \cup \sigma(fr(\varphi))|$, thus from the fact that $|Y'| \geq |adom(s) \cup \sigma(fr(\varphi))| + N_{Ag}$, we have that $|Y' \setminus Im(\gamma)| \geq |Dom(\iota')|$, which guarantees the existence of ι'.

Next, define $j : Dom(j) \mapsto Y'$ as follows:

$$j(u) = \begin{cases} \gamma(u), \text{ if } u \in Dom(\gamma) \\ \iota'(u), \text{ if } u \in Dom(\iota') \end{cases}$$

Clearly, j is invertible. In particular, j is a witness for $s \oplus t \simeq s' \oplus t'$, for $t' = j(t)$. In particular, since $t \in \tau(s, \boldsymbol{\alpha}(\boldsymbol{u}))$, by uniformity we obtain that $t' \in \tau(s', \boldsymbol{\alpha}(j(\boldsymbol{u})))$. Thus, $s' \to t'$. Finally, by construction of t', σ and σ' are equivalent for φ w.r.t. t and t'. □

The proof of Lemma 2 relies crucially on $\mathcal{P}$ and $\mathcal{P}'$ being uniform. Moreover, since $\mathcal{P}$ and $\mathcal{P}'$ are defined on the same P-AbDaS Ag, we do not need to assume that $\mathcal{P}$ and $\mathcal{P}'$ are bisimilar, as it is the case in [7, Lemma 3.9] for instance.

Then, Lemma 2 generalises to runs.

Lemma 3. *Consider C-AbDaS $\mathcal{P}$ and $\mathcal{P}'$ defined on the same P-AbDaS Ag, isomorphic states $s \in S$ and $s' \in S'$, an FO-CTL formula φ, and two assignments σ and σ' equivalent for φ w.r.t. s and s'. For every run r of $\mathcal{P}$, if $r(0) = s$ and for all $i \geq 0$, $|Y'| \geq |adom(r(i)) \cup \sigma(fr(\varphi))| + N_{Ag}$, then there exists a run r' of $\mathcal{P}'$ such that for all $i \geq 0$:*

(i) $r'(0) = s'$;
(ii) $r(i) \simeq r'(i)$;
(iii) *σ and σ' are equivalent for φ w.r.t. $r(i)$ and $r'(i)$.*

Proof. Let r be a run satisfying the lemma's hypothesis. We inductively build r' and show that the conditions *(i)-(iii)* are satisfied. For $i = 0$, let $r'(0) = s'$. By hypothesis, r is such that $|Y'| \geq |adom(r(0)) \cup \sigma(fr(\varphi))| + N_{Ag}$. Thus, since $r(0) \rightarrow r(1)$, by Lemma 2 there exists $t' \in S'$ such that $r'(0) \rightarrow t'$, $r(1) \simeq t'$, and σ and σ' are equivalent for φ w.r.t. $r(1)$ and t'. Let $r'(1) = t'$.

The case for $i > 0$ is similar. Assume that $r(i) \simeq r'(i)$ and σ and σ' are equivalent for φ w.r.t. $r(i)$ and $r'(i)$. Since $r(i) \rightarrow r(i+1)$ and $|Y'| \geq |adom(r(i)) \cup \sigma(fr(\varphi))| + N_{Ag}$, by Lemma 2 there exists $t' \in S'$ such that $r'(i) \rightarrow t'$, σ and σ' are equivalent for φ w.r.t. $r(i+1)$ and t', and $r(i+1) \simeq t'$. Let $r'(i+1) = t'$. It is clear that r' is a run in $\mathcal{P}'$.

Again, Lemma 3 differs from similar results in the literature (e.g., [7, Lemma 3.10]) as we do not need to assume that $\mathcal{P}$ and $\mathcal{P}'$ are bisimilar.

By Lemma 3 we can prove that, for sufficiently large domains, FO-CTL formulas cannot distinguish isomorphic C-AbDaS built on the same P-AbDaS.

Theorem 1. *Consider C-AbDaS $\mathcal{P}$ and $\mathcal{P}'$ defined on the same P-AbDaS Ag, isomorphic states $s \in S$ and $s' \in S'$, an FO-CTL formula φ, and two assignments σ and σ' equivalent for φ w.r.t. s and s'. If*

1. *for every run r such that $r(0) = s$, for all $k \geq 0$ we have $|Y'| \geq |adom(r(k)) \cup \sigma(fr(\varphi))| + |var(\varphi) \setminus fr(\varphi)| + N_{Ag}$;*
2. *for every run r' such that $r'(0) = s'$, for all $k \geq 0$ we have $|Y| \geq |adom(r'(k)) \cup \sigma'(fr(\varphi))| + |var(\varphi) \setminus fr(\varphi)| + N_{Ag}$;*

then $(\mathcal{P}, s, \sigma) \models \varphi$ iff $(\mathcal{P}', s', \sigma') \models \varphi$.

Proof. The proof is by induction on the structure of φ. We prove that if $(\mathcal{P}, s, \sigma) \models \varphi$ then $(\mathcal{P}', s', \sigma') \models \varphi$. The other direction can be proved analogously. The base case for atomic formulas follows by Prop. 1. The inductive cases for propositional connectives are immediate and thus omitted.

For $\varphi \equiv \forall x \psi$, assume that $x \in fr(\psi)$ (otherwise consider ψ, and the corresponding case), and no variable is quantified more than once (otherwise we can rename variables w.l.o.g.). Let γ be a bijection witnessing that σ and σ' are equivalent for φ w.r.t. s and s'. For $u \in adom(s)$, consider the assignment σ^x_u. By definition, $\gamma(u) \in adom(s')$, and $\sigma'^x_{\gamma(u)}$ is well-defined. Note that $fr(\psi) = fr(\varphi) \cup \{x\}$; so σ^x_u and $\sigma'^x_{\gamma(u)}$ are equivalent for ψ w.r.t. s and s'. Moreover, $|\sigma^x_u(fr(\psi))| = |\sigma(fr(\varphi))| + 1$. The same considerations apply to σ'. Further, $|var(\psi) \setminus fr(\psi)| = |var(\varphi) \setminus fr(\varphi)| - 1$, as $var(\psi) = var(\varphi)$,

$fr(\psi) = fr(\varphi) \cup \{x\}$, and $x \notin fr(\varphi)$. Thus, both hypotheses (1) and (2) remain satisfied if we replace φ with ψ, σ with σ^x_u, and σ' with $\sigma'^x_{\gamma(u)}$. Therefore, by the induction hypothesis, if $(\mathcal{P}, s, \sigma^x_u) \models \psi$ then $(\mathcal{P}', s', \sigma'^x_{\gamma(u)}) \models \psi$. Since $u \in \mathit{adom}(s)$ is generic and γ is a bijection, the result follows.

For $\varphi \equiv AX\psi$, assume by contraposition that $(\mathcal{P}', s', \sigma') \not\models \varphi$. Then, there exists a run r' such that $r'(0) = s'$ and $(\mathcal{P}', r'(1), \sigma') \not\models \psi$. Since $|var(\varphi) \setminus fr(\varphi)| \geq 0$, by Lemma 3, there exists a run r such that $r(0) = s$, and for all $i \geq 0$, $r(i) \simeq r'(i)$ and σ and σ' are equivalent for ψ w.r.t. $r(i)$ and $r'(i)$. Since r is a run such that $r(0) = s$, it satisfies hypothesis (1). Moreover, the same hypothesis is necessarily satisfied by all the runs r'' such that for some $i \geq 0$, $r''(0) = r(i)$ (otherwise, the run $r(0) \to \cdots \to r(i) \to r''(1) \to r''(2) \to \cdots$ would not satisfy the hypothesis for r); the same considerations apply w.r.t hypothesis (2) and for all the runs r''' such that $r'''(0) = r'(i)$, for some $i \geq 0$. In particular, these hold for $i = 1$. Thus, we can inductively apply the hypothesis, by replacing s with $r(1)$, s' with $r'(1)$, and φ with ψ (observe that $var(\varphi) = var(\psi)$ and $fr(\varphi) = fr(\psi)$). But then we obtain $(\mathcal{P}, r(1), \sigma) \not\models \psi$, thus $(\mathcal{P}, r(0), \sigma) \not\models AX\psi$.

For $\varphi \equiv E\psi U\phi$, assume that the only variables common to ψ and ϕ occur free in both formulas (otherwise rename quantified variables w.l.o.g.). Let r be a run such that $r(0) = s$, and there exists $k \geq 0$ such that $(\mathcal{P}, r(k), \sigma) \models \phi$, and $(\mathcal{P}, r(j), \sigma) \models \psi$ for $0 \leq j < k$. By Lemma 3 there exists a run r' such that $r'(0) = s'$ and for all $i \geq 0$, $r'(i) \simeq r(i)$ and σ and σ' are equivalent for φ w.r.t. $r'(i)$ and $r(i)$. From each bijection γ_i witnessing that σ and σ' are equivalent for φ w.r.t. $r'(i)$ and $r(i)$, define the bijections $\gamma_{i,\psi} = \gamma_i|_{\mathit{adom}(r(i)) \cup \sigma(fr(\psi))}$ and $\gamma_{i,\phi} = \gamma_i|_{\mathit{adom}(r(i)) \cup \sigma(fr(\phi))}$. Since $fr(\psi) \subseteq fr(\varphi)$, $fr(\phi) \subseteq fr(\varphi)$, it can be seen that $\gamma_{i,\psi}$ and $\gamma_{i,\phi}$ witness that σ and σ' are equivalent for respectively ψ and ϕ w.r.t. $r'(i)$ and $r(i)$. By the same argument used for the AX case above, hypothesis (1) holds for all the runs r'' such that $r''(0) = r(i)$, for some $i \geq 0$, and hypothesis (2) holds for all the runs r''' such that $r'''(0) = r'(i)$. Now observe that $|\sigma(fr(\phi))|, |\sigma(fr(\psi))| \leq |\sigma(fr(\varphi))|$. Moreover, by the assumption on the common variables of ψ and ϕ, $(var(\varphi) \setminus fr(\varphi)) = (var(\psi) \setminus fr(\psi)) \uplus (var(\phi) \setminus fr(\phi))$, thus $|var(\varphi) \setminus fr(\varphi)| = |(var(\psi) \setminus fr(\psi)| + |(var(\phi) \setminus fr(\phi)|$, hence $|(var(\psi) \setminus fr(\psi)|, |(var(\phi) \setminus fr(\phi)| \leq |var(\varphi) \setminus fr(\varphi)|$. Therefore hypotheses (1) and (2) hold also with φ uniformly replaced by either ψ or ϕ. Then, the induction hypothesis applies for each i, by replacing s with $r(i)$, s' with $r'(i)$, and φ with either ψ or ϕ. Thus, for each i, $(\mathcal{P}, r(i), \sigma) \models \psi$ iff $(\mathcal{P}', r'(i), \sigma') \models \psi$, and $(\mathcal{P}, r(i), \sigma) \models \phi$ iff $(\mathcal{P}', r'(i), \sigma') \models \phi$. Therefore, r' is a run such that $r'(0) = s'$, $(\mathcal{P}', r'(k), \sigma') \models \phi$, and for every j, $0 \leq j < k$ implies $(\mathcal{P}', r'(j), \sigma') \models \psi$, i.e., $(\mathcal{P}', s', \sigma') \models E\psi U\phi$.

For $\varphi \equiv A\psi U\phi$, assume by contraposition that $(\mathcal{P}', s', \sigma') \not\models \varphi$. Then, there exists a run r' such that $r'(0) = s'$ and for every $k \geq 0$, either $(\mathcal{P}', r'(k), \sigma') \not\models \phi$ or there exists j such that $0 \leq j < k$ and $(\mathcal{P}', r'(j), \sigma') \not\models \psi$. By Lemma 3 there exists a run r such that $r(0) = s$, and for all $i \geq 0$, $r(i) \simeq r'(i)$ and σ and σ' are equivalent for φ w.r.t. $r(i)$ and $r'(i)$. Similarly to the case of $E\psi U\phi$, it can be shown that σ and σ' are equivalent for ψ and ϕ w.r.t. $r(i)$ and $r'(i)$, for all $i \geq 0$. Further, assuming w.l.o.g. that all variables common to ψ and ϕ occur free in both formulas, it can be shown, as in the case of $E\psi U\phi$, that the induction hypothesis holds on every pair of runs obtained as suffixes of r and r', starting from their i-th state, for every $i \geq 0$. Thus,

$(\mathcal{P}, r(i), \sigma) \models \psi$ iff $(\mathcal{P}', r'(i), \sigma') \models \psi$, and $(\mathcal{P}, r(i), \sigma) \models \phi$ iff $(\mathcal{P}', r'(i), \sigma') \models \phi$. But then r is such that $r(0) = s$ and for every $k \geq 0$, either $(\mathcal{P}, r(k), \sigma) \not\models \phi$ or there exists j such that $0 \leq j < k$ and $(\mathcal{P}, r(j), \sigma) \not\models \psi$, that is, $(\mathcal{P}, s, \sigma) \not\models A\psi U\phi$.

We can now immediately extend Theorem 1 to the model checking problem for C-AbDaS.

Theorem 2. *Consider C-AbDaS $\mathcal{P}$ and $\mathcal{P}'$ defined on the same P-AbDaS Ag, and an FO-CTL formula φ. If*

1. $|Y'| \geq \max_{s \in S} |adom(s)| + |var(\varphi)| + N_{Ag}$;
2. $|Y| \geq \max_{s' \in S'} |adom(s')| + |var(\varphi)| + N_{Ag}$;

then $\mathcal{P} \models \varphi$ iff $\mathcal{P}' \models \varphi$.

Proof. Equivalently, we prove that if $(\mathcal{P}, s_0, \sigma) \not\models \varphi$ for some σ, then there exists a σ' s.t. $(\mathcal{P}', s'_0, \sigma') \not\models \varphi$, and viceversa. To this end, observe that hypotheses (1) and (2) imply, respectively, hypotheses (1) and (2) of Theorem 1. Further, notice that, by cardinality considerations, given the assignment $\sigma : T \mapsto Y$, there exists an assignment $\sigma' : T \mapsto Y'$ such that σ and σ' are equivalent for φ w.r.t. s_0 and s'_0. Thus, by applying Theorem 1 we have that if there exists an assignment σ such that $(\mathcal{P}, s_0, \sigma) \not\models \varphi$, then there exists an assignment σ' such that $(\mathcal{P}', s'_0, \sigma') \not\models \varphi$. The converse can be proved analogously, as the hypotheses are symmetric.

Theorem 2 shows that P-AbDaS Ag can in principle be verified by assuming an interpretation domain of suitable size. Notice again that, since $\mathcal{P}$ and $\mathcal{P}'$ are defined on the same P-AbDaS Ag, differently from [7] we do not require any notion of bisimulation. Moreover, if we are able to bound the quantity $\max_{s \in S} |adom(s)|$ across Ag, then we obtain a cut-off value. These considerations motivate the following definition.

Definition 14 (Bounded P-AbDaS). *A P-AbDaS Ag is b-bounded, for $b \in \mathbb{N}$, if for all C-AbDaS $\mathcal{P}$ based on Ag, for all reachable states $s \in S$, $|adom(s)| \leq b$.*

Boundedness can be justified in terms of the underlying implementation of a P-AbDaS. Indeed, in the order-to-cash scenarios it is likely that there is a maximum number of purchase orders that the manufacturer can deal with at any single time. By assuming boundedness, next result follows from Theorem 2.

Theorem 3. *Consider a b-bounded P-AbDaS Ag over an infinite interpretation domain $\mathcal{Y}$. Then, $n = b + k + N_{Ag}$ is a cut-off for all formulas with at most k variables.*

By Theorem 3 to decide whether a specification φ is true in a bounded P-AbDaS Ag, we can check the corresponding C-AbDaS $\mathcal{P}$ based on increasingly bigger domains $Y \underset{inf}{\subset} \mathcal{Y}$, until we hit $|Y| = b + var(\varphi) + N_{Ag}$. If formula φ is true in all iteration, we can then conclude that φ is true in Ag.

Discussion. The assumption of boundedness to obtain decidability may appear restrictive. However, notice that in most implementation of data-aware systems, the bound is set by the system's specification in terms of memory. That is, we can safely assume that our system will never contain more than a certain amount of data, however large it

can be, and use this bound to verify properties of interest. Unfortunately, the problem of deciding whether a system is b-bounded, for some $b \in \mathbb{N}$, is undecidable in general. Some restrictions on the specification of actions to obtain bounded systems have been explored in [3].

We conclude this section by elaborating on our running example.

Example 3. Consider again the order-to-cash scenario and suppose that the customer can request at most 5 products for each purchase order and the manufacturer can request at most 10 materials to the supplier. Note that, in principle the number of products could be infinite. Further, the total number of products and the total number of materials are both 20. So, we can fix a bound $b = 5 \cdot 4 + 10 \cdot 4 + 20 \cdot 2 + 20 \cdot 2 = 140$, and notice that the FO-CTL specifications in Example 2 contain at most 6 variables. Then, the value for the cut-off is $n = 146 + N_{Ag}$. Since the maximum number of parameters for the customer and the manufacturer is 2 and for the supplier is 1, then $n = 146 + 5 = 151$ is the total cut-off. As a result, to verify the FO-CTL specifications in Example 2 it is sufficient to model check them on C-AbDaS of domain size $|Y| = 151$.

5 Related Work and Conclusions

Amongst the first contributions to consider the verification of data-aware systems we mention [8,18]. This direction was then developed in [15,12], which apply syntactic restrictions on the system description and the specification language in order to obtain decidability. Closely related to the present contribution are [3,7,10], where sufficient conditions for decidable model checking of data-centric dynamic systems are given. Results on the verification of DaS have also appeared in [13,5,6], and then applied to the monitoring of commitments [27] and plan synthesis [9]. While we acknowledge the contribution of these works, there are two important differences in our approach w.r.t. the state of the art. Firstly, we here considered the parameterised model checking problem, where each system is parametric w.r.t. a *finite*, possibly different, interpretation domain; whereas in the references above each system carries its own infinite interpretation domain. Secondly, because of this technical shift, instead of introducing notions of bisimilarity to obtain finite abstractions [7], we rather explore the existence of cut-offs defined on the same agents as the parameterised AbDaS, but with a finite interpretation domain. We believe that this last problem is more interesting for practical applications because, rather than dealing with an actual infinity of data, data-aware systems usually encompass an unbounded number of elements, which is more naturally modelled as a parameterised model checking problem.

On the subject of parameterised model checking of agent-based systems, recently several methodologies and tools have been proposed [23,22]. These contributions are orthogonal, as while they do not model data-aware systems, they are capable of dealing with an arbitrary number of agents. As regards DaS, a method for the verification of parameterised agent-based systems, each encoded via infinite-state models, was presented in [24]. However, this approach only supports a non-quantified specification language and does not deal with (semi-)structured data as we do here. Finally, [4] reports on some preliminaries results on the verification of data-aware multi-agent systems. But

decidability results are available only for a rather limited fragment of the specification language considered therein. The present contribution differs from the works above as, to the best of our knowledge, we introduce for the first time the problem of parameterised model checking for data-aware systems. As we motivated, this is a relevant question for verification, as we aim at guaranteeing the correct behaviour of data-aware systems no matter what the underlying data content is. To this end, we proved theoretical results on the preservation of specifications written in FO-CTL under cardinality constraints. Finally, we showed that such results guarantee the existence of a cut-off for the class of bounded P-AbDaS. We illustrate the relevance of the formal machinery through an application to an IBM use-case, the order-to-cash scenario.

We plan to extend the present work in several directions, including more expressive specification languages, possibly with some form of arithmetic, which is essential for real-life applications. Also of interest are the results in [23,22] that allow for the verification of systems with an arbitrary number of agents. We plan to explore such an extension of our present setting.

Acknowledgements. F. Belardinelli acknowledges the support of ANR JCJC Project SVeDaS (ANR-16-CE40-0021).

References

1. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley (1995).
2. Alur, R., Henzinger, T.: Reactive modules. Formal Methods in System Design **15**(1), 7–48 (1999).
3. Bagheri, B., Calvanese, D., Montali, M., Giacomo, G., Deutsch, A.: Verification of relational data-centric dynamic systems with external services. In: PODS13. pp. 163–174. ACM (2013)
4. Belardinelli, F., Kouvaros, P., Lomuscio, A.: Parameterised verification of data-aware multi-agent systems. In: Proceedings of IJCAI (2017)
5. Belardinelli, F., Lomuscio, A., Patrizi, F.: An abstraction technique for the verification of artifact-centric systems. In: KR12. pp. 319–328 (2012)
6. Belardinelli, F., Lomuscio, A., Patrizi, F.: Verification of GSM-based artifact-centric systems through finite abstraction. In: ICSOC'12. pp. 17–31 (2012)
7. Belardinelli, F., Lomuscio, A., Patrizi, F.: Verification of agent-based artifact systems. J. Artif. Intell. Res. **51**, 333–376 (2014).
8. Bhattacharya, K., Gerede, C.E., Hull, R., Liu, R., Su, J.: Towards Formal Analysis of Artifact-Centric Business Process Models. In: Proc. of BPM (2007)
9. Calvanese, D., Montali, M., Patrizi, F., Stawowy, M.: Plan synthesis for knowledge and action bases. In: IJCAI16. pp. 1022–1029 (2016)
10. Calvanese, D., De Giacomo, G., Montali, M., Patrizi, F.: First-order mu-calculus over generic transition systems and applications to the situation calculus. Information and Computation **259**(3), 328–347 (2018).
11. Cohn, D., Hull, R.: Business Artifacts: A Data-Centric Approach to Modeling Business Operations and Processes. IEEE Data Eng. Bull. **32**(3), 3–9 (2009)
12. Damaggio, E., Deutsch, A., Vianu, V.: Artifact Systems with Data Dependencies and Arithmetic. ACM TDS **37**(3), 22:1–22:36 (2012)
13. De Giacomo, G., Lespérance, Y., Patrizi, F.: Bounded Situation Calculus Action Theories and Decidable Verification. In: KR'12. pp. 467–477 (2012)
14. De Masellis, R., Lembo, D., Montali, M., Solomakhin, D.: Semantic enrichment of gsm-based artifact-centric models. J. Data Semantics **4**(1), 3–27 (2015).

15. Deutsch, A., Hull, R., Patrizi, F., Vianu, V.: Automatic verification of data-centric business processes. In: ICDT09. pp. 252–267. ACM (2009)
16. Deutsch, A., Sui, L., Vianu, V.: Specification and Verification of Data-Driven Web Applications. J. Comput. Syst. Sci. **73**(3), 442–474 (2007)
17. Fagin, R., Halpern, J., Moses, Y., Vardi, M.: Reasoning About Knowledge. The MIT Press (1995)
18. Gerede, C.E., Su, J.: Specification and Verification of Artifact Behaviors in Business Process Models. In: ICSOC'07. pp. 181–192 (2007)
19. Gonzalez, P., Griesmayer, A., Lomuscio, A.: Verification of GSM-based artifact-centric systems by predicate abstraction. In: ICSOC15. LNCS, vol. 9435, pp. 253–268. Springer (2015)
20. Hull, R.: Artifact-centric business process models: Brief survey of research results and challenges. In: OTM Conferences (2). pp. 1152–1163 (2008)
21. Hull, R., Damaggio, E., De Masellis, R., Fournier, F., Gupta, M., Heath, III, F.T., Hobson, S., Linehan, M., Maradugu, S., Nigam, A., Sukaviriya, P.N., Vaculin, R.: Business artifacts with guard-stage-milestone lifecycles: managing artifact interactions with conditions and events. In: Proceedings of DEBS. pp. 51–62, ACM (2011).
22. Kouvaros, P., Lomuscio, A.: Verifying emergent properties of swarms. In: Proceedings of IJCAI15. pp. 1083–1089. AAAI Press (2015)
23. Kouvaros, P., Lomuscio, A.: Parameterised verification for multi-agent systems. Artificial Intelligence **234**, 152–189 (2016)
24. Kouvaros, P., Lomuscio, A.: Parameterised verification of infinite state multi-agent systems via predicate abstraction. In: Proceedings of AAAI17. pp. 3013–3020. AAAI Press (2017)
25. Lomuscio, A., Michaliszyn, J.: Model checking unbounded artifact-centric systems. In: KR14. pp. 488–497 (2014)
26. Montali, M., Calvanese, D.: Soundness of data-aware, case-centric processes. STTT **18**(5), 535–558 (2016).
27. Montali, M., Calvanese, D., De Giacomo, G.: Verification of data-aware commitment-based multiagent system. In: AAMAS14. pp. 157–164. IFAAMAS (2014)
28. O'Leary, D.E.: Artificial intelligence and big data. IEEE Intelligent Systems **28**, 96–99 (03 2013).
29. Shoham, Y., Leyton-Brown, K.: Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations. Cambridge University Press (2008)
30. Singh, M.P., Huhns, M.N.: Service-Oriented Computing: Semantics, Processes, Agents. Wiley (2005)
31. Wooldridge, M.: Computationally Grounded Theories of Agency. In: Proc. of ICMAS, pp. 13–22. IEEE Press (2000)