Journal of Artificial Intelligence Research 51 (2014) Submitted 04/14; published 10/14

Verification of Agent-Based Artifact Systems

Francesco Belardinelli BELARDINELLI@IBISC.FR
Laboratoire Ibisc, Université d'Evry, France
Alessio Lomuscio A.LOMUSCIO@IMPERIAL.AC.UK
Department of Computing, Imperial College London, UK
Fabio Patrizi FABIO.PATRIZI@DIS.UNIROMA1.IT
Dipartimento di Ingegneria Informatica,
Automatica e Gestionale "A. Ruberti"
Università di Roma "La Sapienza", Italy

Abstract

Artifact systems are a novel paradigm for specifying and implementing business processes described in terms of interacting modules called *artifacts*. Artifacts consist of *data* and *lifecycles*, accounting respectively for the relational structure of the artifacts' states and their possible evolutions over time. In this paper we put forward artifact-centric multi-agent systems, a novel formalisation of artifact systems in the context of multi-agent systems operating on them. Differently from the usual process-based models of services, we give a semantics that explicitly accounts for the data structures on which artifact systems are defined.

We study the model checking problem for artifact-centric multi-agent systems against specifications expressed in a quantified version of temporal-epistemic logic expressing the knowledge of the agents in the exchange. We begin by noting that the problem is undecidable in general. We identify a noteworthy class of systems that admit bisimilar, finite abstractions. It follows that we can verify these systems by investigating their finite abstractions; we also show that the corresponding model checking problem is EXPSPACE-complete. We then introduce artifact-centric programs, compact and declarative representations of the programs governing both the artifact system and the agents. We show that, while these in principle generate infinite-state systems, under natural conditions their verification problem can be solved on finite abstractions that can be effectively computed from the programs. We exemplify the theoretical results here pursued through a mainstream procurement scenario from the artifact systems literature.

1. Introduction

Much of the work in the area of *reasoning about knowledge* involves the development of formal techniques for the representation of epistemic properties of rational actors, or *agents*, in a multi-agent system (MAS). The approaches based on modal logic are often rooted on interpreted systems (Parikh & Ramanujam, 1985), a computationally grounded semantics (Wooldridge, 2000) used for the interpretation of several temporal-epistemic logics. This line of research was thoroughly explored in the 1990s leading to a significant body of work (Fagin, Halpern, Moses, & Vardi, 1995; Meyer & van der Hoek, 1995). A recent topic of interest has been the development of automatic techniques, including model checking (Clarke, Grumberg, & Peled, 1999), for the verification of temporal-epistemic specifications for the autonomous agents in a MAS (Gammie & van der Meyden, 2004; Lomuscio, Qu, & Raimondi, 2009; Kacprzak, Nabialek, Niewiadomski, Penczek, Pólrola, Szreter, Wozna, & Zbrzezny, 2008). This has led to developments in a number of areas traditionally outside artificial intelligence, knowledge representation and MAS, including security (Dechesne & Wang, 2010; Ciobaca, Delaune, & Kremer, 2012), web-services (Lomuscio, Penczek, Solanki, & Szreter, 2011) and cache-coherence protocols in hardware design (Baukus & van der Meyden, 2004). The ambition of the present paper is to offer a similar change of perspective in the area of *artifact systems* (Cohn & Hull, 2009), a growing topic in Service-Oriented Computing (SOC).

Artifacts are structures that "combine data and process in an holistic manner as the basic building block[s]" (Cohn & Hull, 2009) of systems' descriptions. Artifact systems are services constituted by complex workflow schemes based on artifacts which the agents interact with. The data component is given by the relational databases underpinning the artifacts in a system, whereas the workflows are described by "lifecycles" associated with each artifact schema. While in the standard service paradigm services are made public by exposing their process interfaces, in artifact systems both the data structures and the lifecycles are advertised. Services are composed in a "hub" where operations on the artifacts are executed. Implementations of artifact systems, such as the IBM engine BARCELONA (Heath et al., 2013), provide a hub where service choreography and service orchestration (Alonso, Casati, Kuno, & Machiraju, 2004) are carried out.

Artifact systems are beginning to drive new application areas, such as case management systems (Marin, Hull, & Vaculín, 2013). However, we identify two shortcomings in the present state-of-the-art. Firstly, the artifact systems literature (Bhattacharya, Gerede, Hull, Liu, & Su, 2007; Deutsch, Hull, Patrizi, & Vianu, 2009; Hull, 2008; Nooijen, Fahland, & Dongen, 2013) focuses exclusively on the artifacts themselves. While there is obviously a need to model and implement the artifact infrastructure, in order to be able to reason comprehensively about artifact systems, there is also a need to account for the agents implementing the services in the system, as it is normally done in the area of reasoning about services (Baresi, Bianculli, Ghezzi, Guinea, & Spoletini, 2007). Secondly, there is a pressing demand to provide the hub with automatic choreography and orchestration capabilities. It is well-known that choreography techniques can be leveraged on automatic model checking techniques; orchestration can be recast as a synthesis problem, which, in turn, can also benefit from model checking technology. However, while model checking and its applications are relatively well-understood in plain process-based modelling, the presence of data makes these problems much harder and virtually unexplored. Additionally, infinite domains in the underlying databases lead to infinite state-spaces and undecidability of the model checking problem.

The aim of this paper is to make a concerted contribution to both problems above. Firstly, we provide a computationally grounded semantics to systems comprising the artifact infrastructure and the agents operating on it. We use this semantics to interpret a temporal-epistemic language with first-order quantifiers to reason about the evolution of the hub as well as the knowledge of the agents in the presence of evolving, structured data. We observe that the model checking problem for these structures is undecidable in general and analyse a notable decidable fragment. For these we derive finite abstractions to infinite-state artifact systems, thereby presenting a technique for their effective verification. We evaluate this methodology by studying its computational complexity and by demonstrating its use on a well-known scenario from the artifact systems literature.

1.1 Artifact-Centric Systems

Service-oriented computing is concerned with the study and development of distributed applications that can be automatically discovered and composed by means of remote interfaces. A point of distinction over more traditional distributed systems is the interoperability and connectedness of services and the shared format for both data and remote procedure calls. Two technology-independent concepts permeate the service-oriented literature: orchestration and choreography (Alonso et al., 2004; Singh & Huhns, 2005). Orchestration involves the ordering of actions of possibly different services, facilitated by a controller or orchestrator, to achieve a certain overall goal. Choreography concerns the distributed coordination of different actions through publicly observable events to achieve a certain goal. A MAS perspective (Wooldridge, 2001) is known to be particularly helpful in service-oriented computing in that it allows us to ascribe information states and private or common goals to the various services. Under this view the agents of the system implement the services and interact with one another in a shared infrastructure or environment.

A key theoretical problem in SOC is to devise effective mechanisms to verify that service composition is correct against some specification. Techniques based on model checking (Clarke et al., 1999) and synthesis (Berardi, Cheikh, Giacomo, & Patrizi, 2008) have been put forward to solve the composition and orchestration problem for services described and advertised at interface level through finite state machines (Calvanese, De Giacomo, Lenzerini, Mecella, & Patrizi, 2008). More recently, attention has turned to services described by languages such as WS-BPEL (Alves et al., 2007), which provide potentially unbounded variables in the description of the service process. Again, model checking approaches have successfully been used to verify complex service compositions (Bertoli, Pistore, & Traverso, 2010; Lomuscio, Qu, & Solanki, 2012).

While WS-BPEL provides a model for services with variables, the data referenced by them is non-permanent. The area of data-centric workflows (Hull et al., 2009; Nigam & Caswell, 2003) evolved as an attempt to provide support for permanent data, typically present in the form of underlying databases. Although usually abstracted away, permanent data is of central importance to services, which typically query data sources and are driven by the answers they obtain; see, e.g., (Berardi, Calvanese, De Giacomo, Hull, & Mecella, 2005). Therefore, a faithful model of a service behaviour cannot, in general, disregard this component. In response to this, proposals have been made in the workflows and service communities in terms of declarative specifications of *data-centric services* that are advertised for automatic discovery and composition. The artifact-centric approach (Cohn & Hull, 2009) is now one of the leading emerging paradigms in the area. Artifact-centric systems can be presented along four dimensions (Hull, 2008; Hull et al., 2011).

Artifacts are the holders of all structured information available in the system. In a business-oriented scenario this may include purchase orders, invoices, payment records, etc. Artifacts may be created, amended, and destroyed at run time; however, abstract artifact schemas are provided at design time to define the structure of all artifacts to be manipulated in the system. Intuitively, external events cause changes in the system, including in the value of artifact attributes.

The evolution of artifacts is governed by **lifecycles**. These capture the changes that an artifact may go through from creation to deletion. Intuitively, a purchase order may be created, amended and operated on before it is fulfilled and its existence in the system terminated: a lifecycle associated with a purchase order artifact formalises these transitions.

Services are seen as the actors operating on the artifact system. They represent both human and software actors, possibly distributed, that generate events on the artifact system. Some services may "own" artifacts, and some artifacts may be shared by several services. However, not all artifacts, or parts of artifacts, are visible to all services. *Views* and *windows* respectively determine which parts of artifacts and which artifact instances are visible to which service. An artifact *hub* is a system that maintains the artifact system and processes the events generated by the services.

Services generate events on the artifact system according to **associations**. Typically these are declarative descriptions providing the precondition and post-conditions for the generation of events. These generate changes in the artifact system according to the artifact lifecycles. Events are processed by a well-defined semantics (Damaggio, Hull, & Vaculín, 2011; Hull et al., 2011) that governs the sequence of changes an artifact system may undertake upon consumption of an event. Such a semantics, based on the use of *Prerequisite-Antecedent-Consequent* (PAC) rules, ensures acyclicity and full determinism in the updates on the artifact system. GSM is a declarative language that can be used to describe artifact systems. BARCELONA is an engine that executes GSM-based artifact systems (Heath et al., 2013).

The above is a partial and incomplete description of the artifact paradigm. We refer to the literature for more details (Cohn & Hull, 2009; Hull, 2008; Hull et al., 2011).

As it will be clear in the next section, in line with the agent-based approach to services, we will use agent-based concepts to model services. The artifact system will be represented as an environment, constituted by evolving databases, upon which the agents operate; lifecycles and associations will be modelled by local and global transition functions. The model is intended to incorporate all artifact-related concepts including views and windows.

In view of the above in this paper we address the following questions. How can we give a transition-based semantics for artifacts and agents operating on them? What language should we use to specify properties of the agents and the artifacts themselves? Can we verify whether or not an artifact system satisfies certain properties? As this will be shown to be undecidable, can we find suitable fragments on which this question can always be answered? If so, what is the resulting complexity? Can we provide declarative specifications for the agent programs so that these can be verified by model checking? Lastly, can this technique be used on mainstream scenarios from the SOC literature?

1.2 Related Work

As stated above, virtually all current literature on artifact-centric systems focuses on properties and implementations of the artifact systems as such. Little or no attention is given to the actors of the system, whether they are human or artificial agents. A few formal techniques have, however, been put forward to verify the core, non-agent aspects of the system; in the following we briefly compare these to this contribution.

To our knowledge the verification of artifact-centric business processes was first discussed by Bhattacharya et al. (2007), where reachability and deadlocks are phrased in the context of artifact-centric systems and complexity results for the verification problem are given. Even disregarding the agent-related aspects here investigated, the present contribution differs markedly from their work by employing a more expressive specification language and by putting forward effective abstraction procedures for verification.

Gerede and Su (2007) study a verification technique for artifact-centric systems against a variant of computation-tree logic. The decidability of the verification problem is proven for the language considered under the assumption that the interpretation domain is bounded. Decidability is also shown for the unbounded case by making restrictions on the values that quantified variables can range over. In the work here presented we also work on unbounded domains, but do not require the restrictions present therein: we only insist on the fact that the number of distinct values in the system does not exceed a given threshold at any point in any run. Most importantly, the interplay between quantification and modalities here considered allows us to bind and use variables in different states. This is a major difference as this feature is very expressive and known to lead potentially to undecidability.

In a related line of research the verification problem for artifact systems against two variants of first-order linear-time temporal logic is considered (Deutsch et al., 2009; Damaggio, Deutsch, & Vianu, 2012). Decidability of the verification problem is retained by imposing syntactic restrictions on both the system description and the specification to check. This effectively limits the way in which new values introduced at every computational step can be used by the system. Properties based on arithmetic operators are also considered (Damaggio et al., 2012). While there are elements of similarity between these approaches and the one we put forward here, including the fact that the concrete interpretation domain is replaced by an abstract one, there are also significant differences. Firstly, our setting is branching-time and not linear-time thereby resulting in different expressive power. Secondly and most importantly, differently from similar contributions (Deutsch et al., 2009; Damaggio et al., 2012), we impose no constraints on nested quantifiers or on their interaction with temporal modalities. In contrast, Damaggio et al. admit only a guarded form of quantification on state formulas, and universal quantification at the outermost syntactic level of the formula, over the free variables of state formulas. These two restrictions represent a major, crucial difference with respect to the present work, in that the former syntactical restrictions prevent representing the interaction between data at different states, which is instead expressible in the present work. Our branching time setting also requires a different abstraction technique. Indeed, the approach above (Deutsch et al., 2009; Damaggio et al., 2012) is based on the construction of a counterexample to the formula to be checked, a fact that is technically made possible by two key factors: (i) the exclusive use of universal quantification over paths, guaranteed by the use of linear-time logics; (ii) the syntactic restriction on quantifiers over values, which permits only universal quantifiers to include temporal modalities within their scope. None of such features is required in our work. Namely, we allow both existential and universal quantification over paths to be present (although in a CTL fashion), and we do not put any restriction on the use of first-order quantifiers. Additionally, the abstraction results we present here are given in general terms on the semantics of declarative programs and do not depend on a particular presentation of the system.

Finally, following an approach similar to ours, Bagheri Hariri et al. (2013) give conditions for the decidability of the model checking problem for data-centric dynamic systems, i.e., dynamic systems with relational states. In this case the specification language used is a first-order version of the μ-calculus. While our temporal fragment is subsumed by the μ-calculus, the two specification languages have different expressive power, since we use indexed epistemic modalities as well as a common knowledge operator. To retain decidability, like we do here, the authors assume a constraint on the size of the states. However, differently from this contribution, Bagheri Hariri et al. also assume limited forms of quantification whereby only individuals persisting in the system evolution can be quantified over. We do not make this restriction here.

Irrespective of what above, the most important feature that characterises our work is that the set-up is entirely based on epistemic logic and multi-agent systems. We use agents to represent the autonomous services operating in the system and agent-based concepts play a key role in the modelling, the specifications, and the verification techniques put forward. Differently from all approaches presented above we are not only concerned with whether the artifact system meets a particular specification. Instead, we also wish to consider what knowledge the agents in the system acquire by interacting among themselves and with the artifact system during a system run.

Additionally, the abstraction methodology put forward is modular with respect to the agents in the system, that is, first we define abstract agents and then compose them together to obtain the abstract system. These features enable us to give constructive procedures for the generation of finite abstractions for artifact-centric programs associated with infinite models. We are not aware of any work in the literature tackling any of these aspects.

This paper combines and expands our preliminary results on artifact-centric systems (Belardinelli, Lomuscio, & Patrizi, 2011a, 2011b, 2012a, 2012b). In particular, the technical set up of artifacts and agents is different from that of our preliminary studies and makes it more natural to express artifact-centric concepts such as views. Differently from our previous attempts, we here incorporate an operator for common knowledge and provide constructive methods to define abstractions. We also consider the complexity of the verification problem, previously unexplored, and evaluate the technique in detail on a case study.

1.3 Scheme of the Paper

The rest of the paper is organised as follows. In Section 2 we introduce artifact-centric multi-agent systems (AC-MAS), the semantics we will be using throughout the paper to describe agents operating on an artifact system. In the same section we put forward FO-CTLK, a first-order logic with knowledge and time to reason about the evolution of the knowledge of the agents and the artifact system. This enables us to propose a satisfaction relation based on the notion of bounded quantification, define the model checking problem, and highlight some properties of isomorphic states. An immediate result we will explore concerns the undecidability of the model checking problem for AC-MAS in their general setting.

Section 3 is devoted to identifying a subclass of AC-MAS that admits a decidable model checking problem against full FO-CTLK specifications. The key finding here is that bounded and uniform AC-MAS, a class identified by studying a strong bisimulation relation, admit finite, truth-preserving abstractions for any FO-CTLK specification. In Section 3.4 we explore the verification problem further and also investigate its complexity thereby showing it to be EXPSPACE-complete.

We turn our attention to artifact programs in Section 4 by defining the concept of artifact-centric programs. We define them through natural, first-order preconditions and post-conditions in line with the artifact-centric approach. We give a semantics to them in terms of AC-MAS and show that their generated models are precisely those uniform AC-MAS studied earlier in the paper. It follows that, under some boundedness conditions which can be naturally expressed, the model checking problem for artifact-centric programs is decidable and can be executed on finite models.

Section 4.2 reports a scenario from the artifact systems literature. This is used to exemplify the technique by providing finite abstractions that can be effectively verified. We conclude in Section 5 where we consider the limitations of the approach and point to further work.

2. Artifact-Centric Multi-agent Systems

In this section we formalise artifact-centric systems and state their verification problem. As data and databases are equally important constituents of artifact systems, our formalisation of artifacts relies on them as underpinning concepts. However, as discussed in the previous section, we here give prominence to agent-based concepts. As such, we define our systems as comprising both the artifacts and the agents interacting with it.

A standard paradigm for logic-based reasoning about agents is *interpreted systems* (Parikh & Ramanujam, 1985; Fagin et al., 1995). In this setting agents are endowed with private local states and evolve by performing actions according to an individual protocol. As data play a key part, as well as to allow us to specify properties of the artifact system, we will define the agents' local states as evolving database instances. We call this formalisation artifact-centric multi-agent systems (AC-MAS). AC-MAS enable us to represent naturally and concisely concepts much used in the artifact paradigm such as the one of *view* discussed earlier.

Our specification language will include temporal-epistemic logic but also quantification over a domain so as to represent the data. This is an usual verification setting, so we will formally define the model checking problem for this set up.

2.1 Databases and First-Order Logic

As discussed above, we use databases as the basic building blocks for defining the states of the agents and the artifact system. We here fix the notation and terminology used. We refer to the literature for more details on databases (Abiteboul, Hull, & Vianu, 1995).

Definition 2.1 (Database Schemas) *A* (relational) database schema $\mathcal{D}$ *is a set* $\{P_1/q_1, \ldots, P_n/q_n\}$ *of* relation symbols P_i*, each associated with its arity* $q_i \in \mathbb{N}$.

Instances of database schemas are defined over interpretation domains, i.e., sets of individuals.

Definition 2.2 (Database Instances) *Given a countable interpretation domain* U *and a database schema* $\mathcal{D}$*, a* $\mathcal{D}$-instance *over* U *is a mapping* D *associating each relation symbol* $P_i \in \mathcal{D}$ *with a* finite q_i*-ary relation over* U*, i.e.,* $D(P_i) \subseteq U^{q_i}$.

The set of all $\mathcal{D}$-instances over a countable interpretation domain U is denoted by $\mathcal{D}(U)$. We simply refer to "instances" whenever the database schema $\mathcal{D}$ is clear by the context. The *active domain* of an instance D, denoted as *adom*(D), is the set of all individuals in U occurring in some tuple of some predicate interpretation $D(P_i)$. Observe that, since $\mathcal{D}$ contains a finite number of relation symbols and each $D(P_i)$ is finite, so is *adom*(D). Also, in the rest of the paper we assume that the interpretation domains are always countable without explictly mentioning this fact.

To fix the notation, we recall the syntax of first-order formulas with equality and no function symbols. Let *Var* be a countable set of *individual variables* and *Con* be a finite set of *individual constants*. A *term* is any element $t \in \mathit{Var} \cup \mathit{Con}$.

Definition 2.3 (FO-formulas over $\mathcal{D}$) *Given a database schema* $\mathcal{D}$*, the formulas* φ *of the first-order language* $\mathcal{L}_\mathcal{D}$ *are defined by the following BNF grammar:*

$$\varphi \quad ::= \quad t = t' \mid P_i(t_1, \ldots, t_{q_i}) \mid \neg\varphi \mid \varphi \rightarrow \varphi \mid \forall x \varphi$$

where $P_i \in \mathcal{D}$*,* $t_1, \ldots, t_{q_i}$ *is a* q_i*-tuple of terms and* t, t' *are terms.*

We assume "$=$" to be a special binary predicate with fixed obvious interpretation. To summarise, $\mathcal{L}_\mathcal{D}$ is a first-order language with equality over the relational vocabulary $\mathcal{D}$ with no function symbols and with finitely many constant symbols from *Con*. Observe that considering a finite set of constants is not a limitation. Indeed, since we will be working with finite sets of formulas, *Con* can always be defined so as to be able to express any formula of interest.

In the following we use the standard abbreviations $\exists$, $\wedge$, $\vee$, and $\neq$. Also, free and bound variables are defined as standard. For a formula φ we denote the set of its variables as $var(\varphi)$, the set of its free variables as $free(\varphi)$, and the set of its constants as $con(\varphi)$. We write $\varphi(\vec{x})$ to list explicitly in arbitrary order all the free variables $x_1, \ldots, x_\ell$ of φ. By slight abuse of notation, we treat $\vec{x}$ as a set, thus we write $\vec{x} = free(\varphi)$. A *sentence* is a formula with no free variables.

Given an interpretation domain U such that $Con \subseteq U$, an *assignment* is a function $\sigma : Var \mapsto U$. For an assignment σ, we denote by $\sigma\binom{x}{u}$ the assignment such that: (i) $\sigma\binom{x}{u}(x) = u$; and (ii) $\sigma\binom{x}{u}(x') = \sigma(x')$, for every $x' \in Var$ different from x. For convenience, we extend assignments to constants so that $\sigma(t) = t$, if $t \in Con$; that is, we assume a Herbrand interpretation of constants. We can now define the semantics of $\mathcal{L}_\mathcal{D}$.

Definition 2.4 (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, *written $(D, \sigma) \models \varphi$, as follows:*

$(D, \sigma) \models P_i(t_1, \ldots, t_{q_i})$	*iff*	$\langle \sigma(t_1), \ldots, \sigma(t_{q_i}) \rangle \in D(P_i)$
$(D, \sigma) \models t = t'$	*iff*	$\sigma(t) = \sigma(t')$
$(D, \sigma) \models \neg\varphi$	*iff*	*it is not the case that* $(D, \sigma) \models \varphi$
$(D, \sigma) \models \varphi \rightarrow \psi$	*iff*	$(D, \sigma) \models \neg\varphi$ *or* $(D, \sigma) \models \psi$
$(D, \sigma) \models \forall x \varphi$	*iff*	*for all* $u \in adom(D)$*, we have that* $(D, \sigma\binom{x}{u}) \models \varphi$

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

Observe that we adopt an *active-domain* semantics, that is, quantified variables range only over the active domain of D. We claim that this form of quantification is sufficient to express specifications of interest (see Section 4.2) while retaining decidability. Also notice that constants are interpreted rigidly; so, two constants are equal if and only if they are syntactically the same. In the rest of the paper, we assume that every interpretation domain includes *Con*. Also, as a usual shortcut, we write $(D, \sigma) \not\models \varphi$ to express that it is not the case that $(D, \sigma) \models \varphi$.

Finally, we introduce the $\oplus$ operator on $\mathcal{D}$-instances that will be used later in the paper. Let the *primed version* of a database schema $\mathcal{D}$ be the schema $\mathcal{D}' = \{P'_1/q_1, \ldots, P'_n/q_n\}$ obtained from $\mathcal{D}$ by syntactically replacing each predicate symbol P_i with its *primed version* P'_i of the same arity.

Definition 2.5 ($\oplus$ Operator) *Given two $\mathcal{D}$-instances D and D', we define $D \oplus D'$ as the $(\mathcal{D} \cup \mathcal{D}')$-instance such that $D \oplus D'(P_i) = D(P_i)$ and $D \oplus D'(P'_i) = D'(P_i)$.*

Intuitively, the $\oplus$ operator defines a disjunctive join of the two instances, where relation symbols in $\mathcal{D}$ are interpreted according to D, while their primed versions are interpreted according to D'.

2.2 Artifact-Centric Multi-agent Systems

In the following we introduce the semantic structures that we will use throughout the paper. We define an artifact-centric multi-agent system as a system comprising an environment representing all artifacts and a finite set of agents interacting with such environment. As agents have views of the artifact state, i.e., projections of the status of particular artifacts, we assume that the building blocks of their private local states are also modelled as database instances. In line with the interpreted systems semantics (Fagin et al., 1995) not everything in the agents' states needs to be present in the environment; a portion of it may be entirely private and not replicated in other agents' states. So, we start by introducing the notion of *agent*.

Definition 2.6 (Agent) *Given an interpretation domain U, an* agent *is a tuple $A = \langle \mathcal{D}, Act, Pr \rangle$, where:*

- *$\mathcal{D}$ is the* local database schema*;*
- *Act is the finite set of* action types *$\alpha(\vec{p})$, where $\vec{p}$ is the tuple of* abstract parameters*;*
- *$Pr : \mathcal{D}(U) \mapsto 2^{Act(U)}$ is the* local protocol function*, where $Act(U)$ is the set of* ground actions *of the form $\alpha(\vec{u})$ where $\alpha(\vec{p}) \in Act$ and $\vec{u} \in U^{|\vec{p}|}$ is a tuple of* ground parameters.

Intuitively, at a given time each agent A is in some local state $l \in \mathcal{D}(U)$ that represents all the information agent A has at its disposal. In this sense we follow the standard approach to multi-agent systems (Fagin et al., 1995), but require that this information is structured as a database. Again, following standard literature we assume that the agents are autonomous and proactive and perform the actions in Act according to the protocol function Pr, which returns the set of grounded actions enabled in any local state. In the definition above we use the term "abstract parameters" to denote variables, i.e., the language in which particular action parameters are given; we use the term "ground parameters" to refer to their concrete values.

We assume that the agents interact among themselves and with an environment comprising all artifacts in the system. As artifacts are entities involving both data and processes, we can see them as collections of database instances paired with actions and governed by special protocols. Without loss of generality we can assume the environment state to be a single database instance including all artifacts in the system. From a purely formal point of view this allows us to represent the environment as a special agent. Of course, in any specific instantiation the environment and the agents will be different entities, exactly in line with the standard propositional version of interpreted systems.

We can therefore define the synchronous composition of agents with the environment.

Definition 2.7 (Artifact-Centric Multi-agent Systems) *Given an interpretation domain U and a set $Ag = \{A_0, \ldots, A_n\}$ of agents $A_i = \langle \mathcal{D}_i, Act_i, Pr_i \rangle$ defined on U, an* artifact-centric multi-agent system *(or AC-MAS) is a tuple $\mathcal{P} = \langle Ag, s_0, \tau \rangle$ where:*

- *$s_0 \in \prod_{A_i \in Ag} \mathcal{D}_i(U)$ is the* initial global state*;*
- *$\tau : \prod_{A_i \in Ag} \mathcal{D}_i(U) \times Act(U) \mapsto 2^{\prod_{A_i \in Ag} \mathcal{D}_i(U)}$ is the* global transition function*, where $Act(U) = Act_0(U) \times \cdots \times Act_n(U)$ is the set of* global (ground) actions*, and $\tau(\langle l_0, \ldots, l_n \rangle, \langle \alpha_0(\vec{u}_0), \ldots, \alpha_n(\vec{u}_n) \rangle)$ is defined whenever $\alpha_i(\vec{u}_i) \in Pr_i(l_i)$ for every $i \leq n$.*

As we will see in later sections, AC-MAS are the natural extension of interpreted systems to the first order to account for environments constituted of artifact-centric systems. They can be seen as a specialisation of quantified interpreted systems (Belardinelli & Lomuscio, 2012), a general extension of interpreted systems to the first-order case.

In the formalisation above the agent A_0 is typically referred to as the *environment E*. The environment normally includes all artifacts in the system (notably by assuming that $\mathcal{D}_0 \supseteq \bigcup_{0<i\leq n} \mathcal{D}_n$), as well as additional information to facilitate communication between the agents and the hub, e.g., messages in transit etc. In what follows we consider $\mathcal{D}_0 = \bigcup_{0<i\leq n} \mathcal{D}_n$ for simplicity; this modelling choice does not impact the results presented later on. At any given time an AC-MAS is described by a tuple of database instances, representing all the agents in the system as well as the artifact

system. A single interpretation domain for all database schemas is given. Note that this does not break the generality of the representation as we can always extend the domain of all agents and the environment before composing them into a single AC-MAS. The global transition function defines the evolution of the system through synchronous composition of actions for the environment and all agents in the system.

Much of the interaction we are interested in modelling involves message exchanges with payload, hence the action parameters, between agents and the environment, i.e., agents operating on the artifacts. However, note that the formalisation above does not preclude us from modelling agent-to-agent interactions, as the global transition function does not rule out successors in which only some agents change their local state following some actions. Also observe that essential concepts such as *views* are easily expressed in AC-MAS by insisting that the local state of an agent includes part of the environment's, i.e., the artifacts the agent has access to. Not all AC-MAS need to have views defined, so it is also possible for the views to be empty.

Other artifact-based concepts such as lifecycles are naturally expressed in AC-MAS. As artifacts are modelled as part of the environment, a lifecycle is naturally encoded in AC-MAS simply as the sequence of changes induced by the transition function τ on the fragment of the environment representing the lifecycle in question. We will show an example of this in Section 2.4.

Some technical remarks now follow. To simplify the notation, we denote a global ground action as $\vec{\alpha}(\vec{u})$, where $\vec{\alpha} = \langle \alpha_0(p_0), \ldots, \alpha_n(p_n) \rangle$ and $\vec{u} = \langle \vec{u}_0, \ldots, \vec{u}_n \rangle$, with each $\vec{u}_i$ of appropriate size. We define the *transition relation* $\rightarrow$ on $\prod_{A_i \in Ag} \mathcal{D}_i(U) \times \prod_{A_i \in Ag} \mathcal{D}_i(U)$ such that $s \rightarrow s'$ if and only if there exists $\vec{\alpha}(\vec{u}) \in Act(U)$ such that $s' \in \tau(s, \vec{\alpha}(\vec{u}))$. If $s \rightarrow s'$, we say that s' is a *successor* of s. A *run* r from $s \in \mathcal{S}$ is an infinite sequence $s^0 \rightarrow s^1 \rightarrow \cdots$, with $s^0 = s$. For $n \in \mathbb{N}$, we take $r(n) \doteq s^n$. A state s' is *reachable from* s if there exists a run r from the global state $r(0) = s$ such that $r(i) = s'$, for some $i \geq 0$. We assume that the relation $\rightarrow$ is serial, i.e., for every global state s there exists s' such that $s \rightarrow s'$. This can be easily obtained by assuming that each agent has a skip action enabled at each local state and that performing skip induces no changes in any of the local states. We introduce $\mathcal{S}$ as the set of states reachable from the initial state s_0 according to the transition relation $\rightarrow$. Notice that assuming a unique initial state does not hinder the generality of the approach, as for any finite set I of states we can encode in τ transitions from s_0 to the states in I. As in plain interpreted systems (Fagin et al., 1995), we say that two global states $s = \langle l_0, \ldots, l_n \rangle$ and $s' = \langle l'_0, \ldots, l'_n \rangle$ are *epistemically indistinguishable* for agent A_i, written $s \sim_i s'$, if $l_i = l'_i$. Differently from interpreted systems the local equality is evaluated on database instances. For convenience we will use also the concept of *temporal-epistemic* (t.e., for short) run. Formally a t.e. run r from a state $s \in \mathcal{S}$ is an infinite sequence $s^0 \rightsquigarrow s^1 \rightsquigarrow \ldots$ such that $s^0 = s$ and $s^i \rightarrow s^{i+1}$ or $s^i \sim_k s^{i+1}$, for some $k \in Ag$. A state s' is said to be *temporally-epistemically reachable* (t.e. reachable, for short) from s if there exists a t.e. run r from the global state $r(0) = s$ such that for some $i \geq 0$ we have that $r(i) = s'$. Obviously, temporal-epistemic runs include purely temporal runs as a special case. Also, notice that we admit U to be infinite, thereby allowing the possibility of the set of states $\mathcal{S}$ to be infinite. Indeed, unless we specify otherwise, we will assume to be working with infinite-state AC-MAS.

Finally, for technical reasons it is useful to refer to the *global* database schema $\mathcal{D} = \mathcal{D}_0 \cup \cdots \cup \mathcal{D}_n$ of an AC-MAS. Every global state $s = \langle l_0, \ldots, l_n \rangle$ is associated with the (global) $\mathcal{D}$-instance $D_s \in \mathcal{D}(U)$ such that $D_s(P_i) = \bigcup_{j \in Ag} l_j(P_i)$, for $P_i \in \mathcal{D}$. We omit the subscript s whenever s is clear from the context and we write *adom*(s) for *adom*(D_s). The justification for this choice comes from the fact that we think of each agent as having a partial, although truthful, view of the global state. If the same relation appears in several agent database schemas, possibly with different interpretations, it means that each agent is aware of only a subset of the total extension of the relation. We maintain that this modeling choice is justified by the application to artifact systems, as it will become apparent in Section 2.4. Notice that for every $s \in \mathcal{S}$, the D_s associated with s is unique, while the converse is not true in general. Finally, we lift the disjoint union operator $\oplus$ to global states so that $s \oplus s' \doteq \langle l_0 \oplus l'_0, \ldots, l_n \oplus l'_n \rangle$. It can be seen that $D_s \oplus D_{s'}$ and $D_{s \oplus s'}$ represent in fact the same $\mathcal{D} \cup \mathcal{D}'$-instance.

2.3 Model Checking

We now define the problem of verifying an artifact-centric multi-agent system against a specification of interest. By following the artifact-centric model, we wish to give data the same prominence as processes. To deal with data and the underlying database instances, our specification language needs to include first-order logic. Further, we require temporal logic to describe the system execution. Lastly, we use epistemic logic to express the information the agents have at their disposal. Hence, we define a first-order temporal-epistemic specification language to be interpreted on AC-MAS. The specification language will be used in Section 4 to formalise properties of artifact-centric programs.

Definition 2.8 (The Logic FO-CTLK) *The first-order CTLK (or FO-CTLK) formulas φ over a database schema $\mathcal{D}$ are inductively defined by the following BNF:*

$$\varphi \ ::= \ \phi \mid \neg\varphi \mid \varphi \rightarrow \varphi \mid \forall x \varphi \mid AX\varphi \mid A\varphi U \varphi \mid E\varphi U \varphi \mid K_i \varphi \mid C\varphi$$

where $\phi \in \mathcal{L}_{\mathcal{D}}$ and $0 < i \leq n$.

The notions of free and bound variables for FO-CTLK extend straightforwardly from $\mathcal{L}_{\mathcal{D}}$, as well as functions *var*, *free*, and *con*. As usual, 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'". The epistemic formulas $K_i\varphi$ and $C\varphi$ intuitively mean that "agent A_i knows φ" and "it is common knowledge among all agents that φ" respectively. We use the abbreviations $EX\varphi$, $AF\varphi$, $AG\varphi$, $EF\varphi$, and $EG\varphi$ as standard. Observe that free variables can occur within the scope of modal operators, thus admitting the unconstrained alternation of quantifiers and modal operators, thereby allowing us to refer to elements in different modal contexts.

The semantics of FO-CTLK formulas is defined as follows.

Definition 2.9 (Satisfaction for FO-CTLK) *Consider an AC-MAS $\mathcal{P}$, an FO-CTLK formula φ, a state $s \in \mathcal{P}$, and an assignment σ. We inductively define whether $\mathcal{P}$* satisfies *φ in s under σ, written $(\mathcal{P}, s, \sigma) \models \varphi$, as follows:*

$(\mathcal{P}, s, \sigma) \models \varphi$	*iff*	$(D_s, \sigma) \models \varphi$, *if φ is an FO-formula*
$(\mathcal{P}, s, \sigma) \models \neg\varphi$	*iff*	*it is not the case that* $(\mathcal{P}, s, \sigma) \models \varphi$
$(\mathcal{P}, s, \sigma) \models \varphi \rightarrow \varphi'$	*iff*	$(\mathcal{P}, s, \sigma) \models \neg\varphi$ *or* $(\mathcal{P}, s, \sigma) \models \varphi'$
$(\mathcal{P}, s, \sigma) \models \forall x \varphi$	*iff*	*for all $u \in adom(s)$,* $(\mathcal{P}, s, \sigma\binom{x}{u}) \models \varphi$
$(\mathcal{P}, s, \sigma) \models AX\varphi$	*iff*	*for all runs 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 runs 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 run 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* $(\mathcal{P}, r(j), \sigma) \models \varphi$
$(\mathcal{P}, s, \sigma) \models K_i\varphi$	*iff*	*for all* $s' \in \mathcal{S}$, $s \sim_i s'$ *implies* $(\mathcal{P}, s', \sigma) \models \varphi$
$(\mathcal{P}, s, \sigma) \models C\varphi$	*iff*	*for all* $s' \in \mathcal{S}$, $s \sim s'$ *implies* $(\mathcal{P}, s', \sigma) \models \varphi$

where $\sim$ *is the transitive closure of* $\bigcup_{1 \leq i \leq n} \sim_i$.

A formula φ is said to be *true* at a state s, written $(\mathcal{P}, s) \models \varphi$, if $(\mathcal{P}, s, \sigma) \models \varphi$ for all assignments σ. Moreover, φ is said to be *true* in $\mathcal{P}$, written $\mathcal{P} \models \varphi$, if $(\mathcal{P}, s_0) \models \varphi$.

A key concern in this paper is to explore the model checking of AC-MAS against first-order temporal-epistemic specifications (Grohe, 2001).

Definition 2.10 (Model Checking Problem) *Given an AC-MAS* $\mathcal{P}$ *and a FO-CTLK formula* φ *the model checking problem consists in finding an assignment* σ *such that* $(\mathcal{P}, s_0, \sigma) \models \varphi$.

It is easy to see that whenever U is finite the model checking problem is decidable as $\mathcal{P}$ is a finite-state system. In general, however, this is not the case. To see this, notice that, by assuming computability, both the agents' protocol functions Pr_i and the AC-MAS transition function τ, can be finitely represented (e.g., as Turing machines). Since all other components of agents and AC-MAS definitions are finite, it follows that all AC-MAS, in particular infinite-state ones, admit a finite representation. Assuming fixed the representation formalism, we have the following result.

Theorem 2.11 *The model checking problem for AC-MAS w.r.t. FO-CTLK is undecidable.*

Proof (sketch). This can be proved by showing that every Turing machine T whose tape contains an initial input I can be simulated by an artifact system $\mathcal{P}_{T,I}$. The problem of checking whether T terminates on that particular input can be reduced to checking whether $\mathcal{P}_{T,I} \models \varphi$, where φ encodes the termination condition. The detailed construction is similar to that of the work of Deutsch, Sui, and Vianu (2007, Thm. 4.10).

□

Given the general setting in which the model checking problem is defined above, the negative result is not surprising. In the following we identify semantic restrictions for which the problem is decidable.

2.4 The Order-to-Cash Scenario

We analyse a business process inspired by a concrete IBM customer use case (Hull et al., 2011). The *order-to-cash* scenario describes the interactions of a number of agents in an e-commerce situation relating to the purchase and delivery of a product. The agents in the artifact-centric system consist of a *manufacturer*, some *customers*, and some *suppliers*. The process begins when a customer prepares and submits a *purchase order* (PO), i.e., a list of products the customer requires, to the manufacturer. Upon receiving a PO, the manufacturer prepares a *material order* (MO), i.e., a list of components needed to assemble the requested products. The manufacturer then selects a supplier and forwards him the relevant material order. Upon receiving an MO a supplier can either accept or reject it. In the former case he then proceeds to deliver the requested components to the manufacturer. In the latter case he notifies the manufacturer of his rejection. If an MO is rejected, the manufacturer deletes it and then prepares and submits a new MO. When the manufacturer receives the delivered components, he assembles the product and, provided the order has been paid

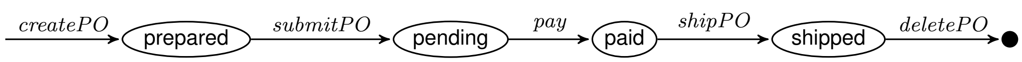
PO			
id	$prod\_code$	$offer$	$status$

MO			
id	$prod\_code$	$price$	$status$

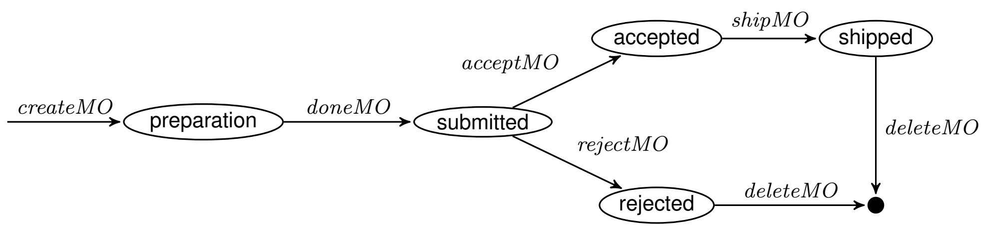
$Products$	
$prod\_code$	$budget$

$Materials$	
mat_code	$cost$

Figure 1: The Data Model for the Order-to-Cash Scenario.



(a) Purchase Order lifecyle



(b) Material Order lifecyle

Figure 2: Lifecycles of the artifacts involved in the order-to-cash scenario.

for, delivers it to the customer. Any manufacturer order which is directly or indirectly related to a PO can be deleted only after the PO is deleted.

We can encode the *order-to-cash* business process as an artifact-centric multi-agent system, where the artifact data models are represented as database schemas and their evolution is characterised by an appropriate set of operations. It is natural to identify two artifact types, representing the PO and the MO. We reserve a distinguished relation for each artifact type. In addition, we introduce *static* relations to store product and material information. As a result, the data model of the order-to-cash scenario with associated attributes can be given as in Figure 1.

The intended meaning of relations is self-explanatory. Note the presence of the attribute $status$ in the relations corresponding to artifact classes. An intuitive representation of the artifact lifecycles, i.e., the evolution of some key records in the artifacts' states, capturing only the dependence of actions from the artifact statuses, is shown in Figure 2. For example, a purchase order, with initial status prepared, is created by the agent customer through the action $createPO$. Once the order is submitted to the agent manufacturer, the PO status changes to pending. The other transitions labelled by pay, $shipPO$ and $deletePO$, act similarly, according to their semantics, on the status of the purchase order. Note that this is an incomplete representation of the business process, as the interaction between actions and the artifact data content is not represented.

We now formally encode the scenario as an AC-MAS. For the sake of presentation in what follows we assume to be dealing with three agents only: one customer c, one manufacturer m and one supplier s. The database schema $\mathcal{D}_i$ for each agent $i \in \{c, m, s\}$ can be given as:

- Customer c:
  $\mathcal{D}_c = \{Products(prod\_code, budget), PO(id, prod\_code, offer, status)\}$;
- Manufacturer m:
  $\mathcal{D}_m = \{PO(id, prod\_code, offer, status), MO(id, prod\_code, price, status)\}$;
- Supplier s:
  $\mathcal{D}_s = \{Materials(mat\_code, cost), MO(id, prod\_code, price, status)\}$.

We consider the infinite set U_{otc} of alphanumeric strings as the interpretation domain, and introduce a parametric action for each transition in the lifecycles in Figure 2. Also, we assume that in the initial state the only non-empty relations are $Products$ and $Materials$, which contain background information, such as a catalogue of available products. We can now define the agents in the order-to-cash scenario as follows.

Definition 2.12 *the agents A_c, A_m and A_s are given as*

- $A_c = \langle \mathcal{D}_c, Act_c, Pr_c \rangle$, *where (i) $\mathcal{D}_c$ is as above; (ii) $Act_c = \{createPO(id, pcode), submitPO(id), pay(id), deletePO(id)\}$; and (iii) Pr_c respects the intended meaning of the customer's actions. For instance, $createPO(id, pcode) \in Pr_c(l_c)$ iff the interpretation $l_c(Products)$ of the relation Products in the local state l_c contains a tuple $\langle pcode, b \rangle$ for some budget b.*
- $A_m = \langle \mathcal{D}_m, Act_m, Pr_m \rangle$, *where (i) $\mathcal{D}_m$ is as above; (ii) $Act_m = \{createMO(id, price), doneMO(id), shipPO(id), deleteMO(id)\}$; and (iii) Pr_m respects the intended meaning of the manufacturer's actions. For instance, $createMO(po\_id, price) \in Pr_m(l_m)$ iff the interpretation $l_m(MO)$ of the relation MO in the local state l_m does not contain a tuple $\langle po\_id, pc, pr, \mathsf{preparation} \rangle$ for the same PO id po_id.*
- $A_s = \langle \mathcal{D}_s, Act_s, Pr_s \rangle$, *where (i) $\mathcal{D}_s$ is as above; (ii) $Act_s = \{acceptMO(id), rejectMO(id), shipMO(id)\}$; and (iii) Pr_c respects the intended meaning of the suppliers' actions. For instance, $acceptMO(mo\_id) \in Pr_s(l_s)$ iff $l_s(MO)$ does not contain a tuple with id mo_id and status* accepted.

Further, we can now define the AC-MAS induced by the set of agents $Ag_{otc} = \{A_c, A_m, A_s\}$ according to Definition 2.7.

Definition 2.13 *Given the set of agents $Ag_{otc} = \{A_c, A_m, A_s\}$, the AC-MAS $\mathcal{P}_{otc} = \langle Ag_{otc}, s^0_{otc}, \tau_{otc} \rangle$ is such that*

- $s^0_{otc} = \langle l_c, l_m, l_s \rangle$ *is the initial global state, where the only non-empty relations are $Products$ and $Materials$ in l_c and l_s respectively;*
- τ_{otc} *is the global transition function defined so as to respect the intended meaning of the evolution of the order-to-cash scenario. For instance, consider the global action*

$\alpha(\vec{u}) = \langle createPO(pc), doneMO(m), acceptMO(m')\rangle$ enabled by the respective protocols in a global state s. By the definition of the actions $createPO(pc)$, $doneMO(m)$, and $acceptMO(m')$ we have that $l_i(s) \in Pr_i$ for $i \in \{c, m, s\}$ implies that the Products relation contains information about the product pc. Also, the interpretation of the relation MO contains the tuples $\langle m, p, pr, \mathsf{preparation}\rangle$ and $\langle m', p', pr', \mathsf{submitted}\rangle$ for some products p and p'. Hence, $s' \in \tau_{otc}(s, \alpha(\vec{u}))$ iff the interpretation of the relation PO in s' extends $D_s(PO)$ with the tuple $\langle id, pc, b, \mathsf{prepared}\rangle$, where id is a fresh id. The tuples for the material orders m and m' are updated in $D_{s'}(MO)$ by becoming $\langle m, p, pr, \mathsf{submitted}\rangle$ and $\langle m', p', pr', \mathsf{accepted}\rangle$, respectively. No other element is changed in the transition.

Clearly, the function τ_{otc} given above can easily be completed to encode the artifacts' lifecycles as given in Figure 2. In section 4.2 we will give a succinct encoding of $\mathcal{P}_{otc}$ in terms of an artifact-centric program.

We can now investigate properties of the present business process by using specifications in FO-CTLK. For instance, the following formula intuitively specifies that the manufacturer m knows that each material order MO has to match a corresponding purchase order PO:

$$\varphi_{match} \quad = \quad AG\, \forall id, pc\, (\exists pr, s\; MO(id, pc, pr, s) \rightarrow K_m \exists o, s' PO(id, pc, o, s'))$$

The next specification states that given a material order MO, the customer will eventually know that the corresponding PO will be shipped.

$$\varphi_{fulfil} \quad = \quad AG\, \forall id, pc\, (\exists pr, s\; MO(id, pc, pr, s) \rightarrow EF\; K_c\; \exists o\; PO(id, pc, o, \mathsf{shipped}))$$

Further, we may be interested in checking whether or not budget and costs are always kept secret from the supplier s and the customer c respectively, and whether the customer (resp., the supplier) knows this fact:

$$\begin{aligned} \varphi_{budget} &= K_c\, \forall pc\; AG\, \neg\exists b\; K_s\; Products(pc, b) \\ \varphi_{cost} &= K_s\, \forall mc\; AG\, \neg\exists c\; K_c\; Materials(mc, c) \end{aligned}$$

Other interesting specifications describing properties of the artifact system and the agents operating in it can be similarly formalised in FO-CTLK, thereby providing the engineer with a valuable tool to assess the implementation.

Observe that the AC-MAS for the order-to-cash scenario has an infinite number of states thereby making it difficult to investigate by means of traditional model checking techniques. We will return to this scenario in Subsection 4.2 where we will investigate how it may still be verified. Before doing so we develop a methodology for associating finite abstractions to infinite AC-MAS.

3. Abstraction for Artifact-Centric Multi-agent Systems

In the previous section we have observed that model checking AC-MAS against FO-CTLK is undecidable in general. It is clearly of interest to isolate decidable settings. In what follows we identify semantic constraints resulting in a decidable model checking problem. The investigation is carried out on a rather natural subclass of AC-MAS that we call *bounded*, as defined below. Our goal for proceeding in this manner is to identify finite abstractions of infinite-state AC-MAS so that verification of programs that admit bounded AC-MAS as models can be conducted on them, rather than on the original infinite-state AC-MAS. We will see this in detail in Section 4.

The key concept that enables us to achieve the above is *uniformity*. Uniform AC-MAS are systems for which the behaviour does not depend on the actual data present in the states. This means that the system contains all possible transitions that are enabled according to parametric action rules, thereby resulting in a "full" transition relation. This notion is related to *genericity* in databases (Abiteboul et al., 1995). Here we use the term "uniformity" as we refer to transition systems and not databases.

To achieve finite abstractions we proceed as follows. We first propose an adaptation of the notion of isomorphism to our setting; then we introduce bisimulations; finally in Subsection 3.2 we show how this notion can be exploited to guarantee that uniform AC-MAS satisfy the same FO-CTLK formulas. We then use this result to show that bounded, uniform systems admit finite abstractions (Subsection 3.3). The complexity of the model checking problem is analysed in Subsection 3.4.

In the rest of the section we let $\mathcal{P} = \langle Ag, s_0, \tau \rangle$ and $\mathcal{P}' = \langle Ag', s'_0, \tau' \rangle$ be two AC-MAS and assume, unless stated differently, that $s = \langle l_0, \ldots, l_n \rangle \in \mathcal{S}$, and $s' = \langle l'_0, \ldots, l'_n \rangle \in \mathcal{S}'$.

3.1 Isomorphisms

We now investigate the concept of isomorphism on AC-MAS. This will be needed in later sections to define finite abstractions of infinite-state AC-MAS.

Definition 3.1 (Isomorphism) *Two local states $l, l' \in \mathcal{D}(U)$ are* isomorphic*, written $l \simeq l'$, iff there exists a bijection $\iota : adom(l) \cup Con \mapsto adom(l') \cup Con$ such that:*

(i) ι is the identity on Con;

(ii) for every $P_i \in \mathcal{D}$, $\vec{u} \in U^{q_i}$, we have that $\vec{u} \in l(P_i)$ iff $\iota(\vec{u}) \in l'(P_i)$.

When this is the case, we say that ι is a witness *for $l \simeq l'$.*

Two global states $s \in \mathcal{S}$ and $s' \in \mathcal{S}'$ are isomorphic*, written $s \simeq s'$, iff there exists a bijection $\iota : adom(s) \cup Con \mapsto adom(s') \cup Con$ such that for every $j \in Ag$, ι is a witness for $l_j \simeq l'_j$.*

Notice that isomorphisms preserve the interpretation of constants in *Con* as well as of predicates in the local states up to renaming of the corresponding terms. Any function ι as above is called a *witness* for $s \simeq s'$. Obviously, the relation $\simeq$ is an equivalence relation. Given a function $f : U \mapsto U'$ defined on $adom(s)$, $f(s)$ denotes the instance in $\mathcal{D}(U')$ obtained from s by renaming each $u \in adom(s)$ as $f(u)$. If f is also injective (thus invertible) and the identity on *Con*, then $f(s) \simeq s$.

Example As an example of isomorphic states, consider an agent with local database schema $\mathcal{D} = \{P_1/2, P_2/1\}$, let $U = \{a, b, c, \ldots\}$ be an interpretation domain, and fix the set $Con = \{b\}$ of constants. Let l be the local state such that $l(P_1) = \{\langle a, b\rangle, \langle b, d\rangle\}$ and $l(P_2) = \{a\}$ (see Figure 3). Then, the local state l' such that $l'(P_1) = \{\langle c, b\rangle, \langle b, e\rangle\}$ and $l'(P_2) = \{c\}$ is isomorphic to l. This can be easily seen by considering the isomorphism ι, where: $\iota(a) = c$, $\iota(b) = b$, and $\iota(d) = e$. However, the state l'' where $l''(P_1) = \{\langle f, d\rangle, \langle d, e\rangle\}$ and $l''(P_2) = \{f\}$ is not isomorphic to l. Indeed, although a bijection exists that maps l into l'', it is easy to see that none can be such that $\iota'(b) = b$.

Note that, while isomorphic states have 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.

$l(P_1)$	
a	b
b	d

$l(P_2)$
a

$l'(P_1)$	
c	b
b	e

$l'(P_2)$
c

$l''(P_1)$	
f	d
d	e

$l''(P_2)$
f

Figure 3: Examples of isomorphic and non-isomorphic local states.

Definition 3.2 (Equivalent assignments) *Given two states $s \in \mathcal{S}$ and $s' \in \mathcal{S}'$, and a set of variables $V \subseteq Var$, two assignments $\sigma : Var \mapsto U$ and $\sigma' : Var \mapsto U'$ are* equivalent for V w.r.t. s and s' *iff there exists a bijection $\gamma : adom(s) \cup Con \cup \sigma(V) \mapsto adom(s') \cup Con \cup \sigma'(V)$ such that:*

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

(ii) $\sigma'|_V = \gamma \circ \sigma|_V$.

Intuitively, equivalent assignments preserve both the (in)equalities of the variables in V and the constants in s, s' up to renaming. Note that, by definition, the above implies that s, s' are isomorphic. We say that two assignments are *equivalent for an FO-CTLK formula* φ, omitting the states s and s' when clear from the context, if these are equivalent for *free*(φ).

We can now show the following standard result in first-order (non-modal) logic, i.e., isomorphic states satisfy exactly the same FO-formulas (Abiteboul et al., 1995).

Proposition 3.3 *Given two isomorphic states $s \in \mathcal{S}$ and $s' \in \mathcal{S}'$, an FO-formula φ, and two assignments σ and σ' equivalent for φ, we have that*

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

Moreover, if ψ is an FO-sentence,

$$D_s \models \psi \quad \textit{iff} \quad D_{s'} \models \psi$$

Thus, isomorphic states cannot be distinguished by FO-sentences. This enables us to use this notion, when defining simulations.

3.2 Bisimulations

Plain bisimulations are known to be satisfaction preserving in a modal propositional setting (Blackburn, de Rijke, & Venema, 2001). In the following we explore the conditions under which this applies to AC-MAS as well. We introduce a notion of bisimulation, based on isomorphisms, and later explore its properties in the context of uniform AC-MAS.

Definition 3.4 (Simulation) *A relation R on $\mathcal{S} \times \mathcal{S}'$ is a* simulation *if $\langle s, s' \rangle \in R$ implies:*

1. $s \simeq s'$;
2. *for every $t \in \mathcal{S}$, if $s \to t$ then there exists $t' \in \mathcal{S}'$ s.t. $s' \to t'$, $s \oplus t \simeq s' \oplus t'$, and $\langle t, t' \rangle \in R$;*
3. *for every $t \in \mathcal{S}$, for every $0 < i \leq n$, if $s \sim_i t$ then there exists $t' \in \mathcal{S}'$ s.t. $t \sim_i t'$, $s \oplus t \simeq s' \oplus t'$, and $\langle t, t' \rangle \in R$.*

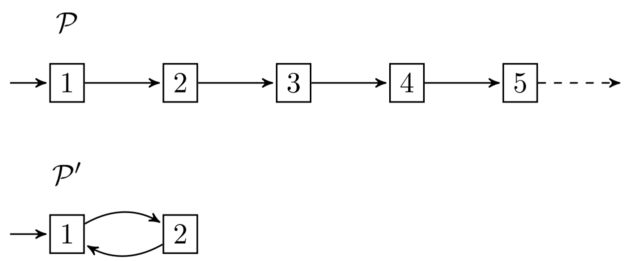


Figure 4: Bisimilar AC-MAS not satisfying the same FO-CTLK formulas.

Definition 3.4 has many similarities with the standard notion of simulation in the propositional setting. In particular, the co-inductive structure of the definition requires similar states to satisfy some local property and to preserve this along corresponding transitions. However, differently from the propositional case, we here insist that $s \oplus t \simeq s' \oplus t'$; this ensures that similar transitions in AC-MAS preserve isomorphic disjoint unions.

A state $s' \in \mathcal{S}'$ is said to *simulate* $s \in \mathcal{S}$, written $s \preceq s'$, iff there exists a simulation R s.t. $\langle s, s' \rangle \in R$. When no ambiguity arises, we simply say that s and s' are *similar*. Note that similar states are isomorphic, as condition (2) above ensures that $s \simeq s'$. The similarity relation can be shown to be the largest simulation, reflexive and transitive on $\mathcal{S} \cup \mathcal{S}'$. Further, we say that $\mathcal{P}'$ *simulates* $\mathcal{P}$, written $\mathcal{P} \preceq \mathcal{P}'$, if $s_0 \preceq s'_0$.

Simulations can naturally be extended to bisimulations, as follows.

Definition 3.5 (Bisimulation) *A relation B on $\mathcal{S} \times \mathcal{S}'$ is a* bisimulation *iff both B and $B^{-1} = \{\langle s', s \rangle \mid \langle s, s' \rangle \in B\}$ are simulations.*

Two states $s \in \mathcal{S}$ and $s' \in \mathcal{S}'$ are said to be bisimilar, written $s \approx s'$, iff there exists a bisimulation B such that $\langle s, s' \rangle \in B$. It can be shown that $\approx$ is the largest bisimulation, and an equivalence relation, on $\mathcal{S} \cup \mathcal{S}'$. We say that $\mathcal{P}$ and $\mathcal{P}'$ are *bisimilar*, written $\mathcal{P} \approx \mathcal{P}'$ iff so are s_0 and s'_0.

It is instructive to note that bisimilar systems do not preserve FO-CTLK formulas. This is markedly different from the modal propositional case.

Example Consider Figure 4, where *Con* $= \emptyset$ and $\mathcal{P}$ and $\mathcal{P}'$ are given as follows. For a number n of agents equal to 1, we define $\mathcal{D} = \mathcal{D}' = \{P/1\}$ and $U = \mathbb{N}$; $s_0(P) = s'_0(P) = \{1\}$; $\tau = \{\langle s, s' \rangle \mid s(P) = \{i\}, s'(P) = \{i+1\}\}$; $\tau' = \{\langle s, s' \rangle \mid s(P) = \{i\}, s'(P) = \{(i \mod 2) + 1\}\}$. Notice that $\mathcal{S} \subseteq \mathcal{D}(\mathbb{N})$ and $\mathcal{S}' \subseteq \mathcal{D}(\mathbb{N})$. Clearly we have that $\mathcal{P} \approx \mathcal{P}'$. Now, consider the constant-free FO-CTLK formula $\varphi = AG(\forall x(P(x) \rightarrow AX\,AG\neg P(x)))$. It can be easily seen that $\mathcal{P} \models \varphi$ while $\mathcal{P}' \not\models \varphi$.

The above shows that, differently from the propositional case, bisimilarity is not a sufficient condition to guarantee the preservation of FO-CTLK formulas. Intuitively, this is a consequence of the fact that bisimilar AC-MAS do not preserve value associations along runs. For instance, the value 1 in $\mathcal{P}'$ is associated infinitely many times with the odd values occurring in $\mathcal{P}$. By quantifying across states we are able to express this fact and can therefore distinguish the two structures. This is a difficulty as, intuitively, we would like to use bisimulations to demonstrate the existence of finite abstractions. However, as we show later, this happens on the class of *uniform* AC-MAS, defined below.

Definition 3.6 (Uniformity) *An AC-MAS $\mathcal{P}$ is said to be* uniform *iff for every $s, t, s' \in \mathcal{S}$, $t' \in \mathcal{D}(U)$,*

1. *if $t \in \tau(s, \vec{\alpha}(\vec{u}))$ and $s \oplus t \simeq s' \oplus t'$ for some witness ι, then for every constant-preserving bijection ι' that extends ι to $\vec{u}$, we have that $t' \in \tau(s', \vec{\alpha}(\iota'(\vec{u})))$;*
2. *if $s \sim_i t$ and $s \oplus t \simeq s' \oplus t'$, then $s' \sim_i t'$.*

This definition captures the idea that actions take into account and operate only on the relational structure of states and action parameters, irrespective of the actual data they contain (apart from a finite set of constants). Intuitively, uniformity expresses that if t can be reached by executing $\alpha(\vec{u})$ in s, and we replace the same element v with v' in s, $\vec{u}$ and t, obtaining s', $\vec{u}'$ and t', then t' can be reached by executing $\alpha(\vec{u}')$ in s'. In terms of the underlying Kripke structures, i.e., the frames induced on $\mathcal{S}$ by relations $\rightarrow$ and $\sim_i$ for $A_i \in Ag$, this means that the systems are "full up to $\oplus$", that is, in all uniform AC-MAS the states t' identified above are indeed part of the system and reachable from s'. A similar condition is required on the epistemic relation. A property of uniform systems is that the latter requirement is implied by the former, as shown by the following result.

Proposition 3.7 *If an AC-MAS $\mathcal{P}$ satisfies req. 1 in Def. 3.6 and $adom(s_0) \subseteq Con$, then req. 2 is also satisfied.*

Proof. If $s \oplus t \simeq s' \oplus t'$, then there is a witness $\iota : adom(s) \cup adom(t) \cup Con \mapsto adom(s') \cup adom(t') \cup Con$ that is the identity on *Con* (hence on $adom(s_0)$). Assume $s \sim_i t$, thus $l_i(s) = l_i(t)$, and $l_i(s') = \iota(l_i(s)) = \iota(l_i(t)) = l_i(t')$. Notice that this does not guarantee that $s' \sim_i t'$, as we need to prove that $t' \in \mathcal{S}$. This can be done by showing that t' is reachable from s_0. Since t is reachable from s_0, there exists a run $s_0 \rightarrow s_1 \rightarrow \ldots \rightarrow s_k$ s.t. $s_k = t$. Extend now ι to a total and injective function $\iota' : adom(s_0) \cup \cdots \cup adom(s_k) \cup Con \mapsto U$. This can always be done because $|U| \geq |adom(s_0) \cup \cdots \cup adom(s_k) \cup Con|$. Now consider the sequence $\iota'(s_0), \iota'(s_1), \ldots, \iota'(s_k)$. Since $adom(s_0) \subseteq Con$ then $\iota(s_0) = s_0$ and, because ι' extends ι, we have that $\iota'(s_0) = \iota(s_0) = s_0$. Further, $\iota'(s_k) = \iota(t) = t'$. By repeated applications of req. 1 we can show that $\iota'(s_{m+1}) \in \tau(\iota'(s_m), \vec{\alpha}(\iota'(\vec{u})))$ whenever $s_{m+1} \in \tau(s_m, \vec{\alpha}(\vec{u}))$, for $m < k$. Hence, the sequence is actually a run from s_0 to t'. Thus, $t' \in \mathcal{S}$, and $s' \sim_i t'$. $\square$

Thus, as long as $adom(s_0) \subseteq Con$, to check whether an AC-MAS is uniform, it is sufficient to take into account only the transition function.

A further distinctive feature of uniform systems is that all isomorphic states are bisimilar.

Proposition 3.8 *If an AC-MAS $\mathcal{P}$ is uniform, then for every $s, s' \in \mathcal{S}$, $s \simeq s'$ implies $s \approx s'$.*

Proof. We prove that $B = \{\langle s, s' \rangle \in \mathcal{S} \times \mathcal{S} \mid s \simeq s'\}$ is a bisimulation. Observe that since $\simeq$ is an equivalence relation, so is B. Thus B is symmetric and $B = B^{-1}$. Therefore, proving that B is a simulation proves also that B^{-1} is a simulation; hence, that B is a bisimulation. To this end, let $\langle s, s' \rangle \in B$, and assume $s \rightarrow t$ for some $t \in \mathcal{S}$. Then, $t \in \tau(s, \alpha(\vec{u}))$ for some $\alpha(\vec{u}) \in Act(U)$. Consider a witness ι for $s \simeq s'$. By cardinality considerations ι can be extended to a total and injective function $\iota' : adom(s) \cup adom(t) \cup \{\vec{u}\} \cup Con \mapsto U$. Consider $\iota'(t) = t'$; it follows that ι' is a witness for $s \oplus t \simeq s' \oplus t'$. Since $\mathcal{P}$ is uniform, $t' \in \tau(s', \alpha(\iota'(\vec{u})))$, that is, $s' \rightarrow t'$. Moreover, ι' is a witness for $t \simeq t'$, thus $\langle t, t' \rangle \in B$. Next assume that $\langle s, s' \rangle \in B$ and $s \sim_i t$, for some $t \in \mathcal{S}$.

By reasoning as above we can find a witness ι for $s \simeq s'$, and an extension ι' of ι s.t. $t' = \iota'(t)$ and ι' is a witness for $s \oplus t \simeq s' \oplus t'$. Since $\mathcal{P}$ is uniform, $s' \sim_i t'$ and $\langle t, t' \rangle \in B$. $\square$

This result intuitively means that submodels generated by isomorphic states are bisimilar.

Next we prove some partial results, which will be useful in proving our main preservation theorem. The first two guarantee that under appropriate cardinality constraints the bisimulation preserves the equivalence of assignments w.r.t. a given FO-CTLK formula.

Lemma 3.9 *Consider two bisimilar and uniform AC-MAS $\mathcal{P}$ and $\mathcal{P}'$, two bisimilar states $s \in \mathcal{S}$ and $s' \in \mathcal{S}'$, and an FO-CTLK formula φ. For every assignments σ and σ' equivalent for φ w.r.t. s and s', we have that:*

1. *for every $t \in \mathcal{S}$ s.t. $s \to t$, if $|U'| \geq |adom(s) \cup adom(t) \cup Con \cup \sigma(free(\varphi))|$, then there exists $t' \in \mathcal{S}'$ s.t. $s' \to t'$, $t \approx t'$, and σ and σ' are equivalent for φ w.r.t. t and t'.*
2. *for every $t \in \mathcal{S}$ s.t. $s \sim_i t$, if $|U'| \geq |adom(s) \cup adom(t) \cup Con \cup \sigma(free(\varphi))|$, then there exists $t' \in \mathcal{S}'$ s.t. $s' \sim_i t'$, $t \approx t'$, and σ and σ' are equivalent for φ w.r.t. t and t'.*

Proof. To prove (1), let γ be a bijection witnessing that σ and σ' are equivalent for φ w.r.t. s and s'. Suppose that $s \to t$. Since $s \approx s'$, by definition of bisimulation there exists $t'' \in \mathcal{S}'$ s.t. $s' \to t''$, $s \oplus t \simeq s' \oplus t''$, and $t \approx t''$. Now define $Dom_j \doteq adom(s) \cup adom(t) \cup Con$, and partition it into:

- $Dom_\gamma \doteq adom(s) \cup Con \cup (adom(t) \cap \sigma(free(\varphi))$;
- $Dom_{\iota'} \doteq adom(t) \setminus Dom_\gamma$.

Let $\iota' : Dom_{\iota'} \mapsto U' \setminus Im(\gamma)$ be an invertible total function. Observe that $|Im(\gamma)| = |adom(s') \cup Con \cup \sigma'(free(\varphi))| = |adom(s) \cup Con \cup \sigma(free(\varphi))|$, thus from the fact that $|U'| \geq |adom(s) \cup adom(t) \cup Con \cup \sigma(free(\varphi))|$ we have $|U' \setminus Im(\gamma)| \geq |Dom(\iota')|$, which guarantees the existence of ι'.

Next, define $j : Dom_j \mapsto U'$ as follows:

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

Obviously, j is invertible. Thus, j is a witness for $s \oplus t \simeq s' \oplus t'$, where $t' = j(t)$. Since $s \oplus t \simeq s' \oplus t''$ and $\simeq$ is an equivalence relation, we obtain that $s' \oplus t' \simeq s' \oplus t''$. Thus, $s' \to t'$, as $\mathcal{P}'$ is uniform. Moreover, σ and σ' are equivalent for φ w.r.t. t and t', by construction of t'. To check that $t \approx t'$, observe that, since $t' \simeq t''$ and $\mathcal{P}'$ is uniform, by Prop. 3.8 it follows that $t' \approx t''$. Thus, since $t \approx t''$ and $\approx$ is transitive, we obtain that $t \approx t'$. The proof for (2) has an analogous structure and therefore is omitted. $\square$

It can be proven that this result is tight, i.e., that if the cardinality requirement is violated, there exist cases where assignment equivalence is not preserved along temporal or epistemic transitions.

Lemma 3.9 easily generalises to t.e. runs.

Lemma 3.10 *Consider two bisimilar and uniform AC-MAS $\mathcal{P}$ and $\mathcal{P}'$, two bisimilar states $s \in \mathcal{S}$ and $s' \in \mathcal{S}'$, an FO-CTLK formula φ, and two assignments σ and σ' equivalent for φ w.r.t. s and s'. For every t.e. run r of $\mathcal{P}$, if $r(0) = s$ and for all $i \geq 0$, $|U'| \geq |adom(r(i)) \cup adom(r(i+1)) \cup Con \cup \sigma(free(\varphi))|$, then there exists a t.e. run r' of $\mathcal{P}'$ s.t. for all $i \geq 0$:*

(i) $r'(0) = s'$*;*

(ii) $r(i) \approx r'(i)$*;*

(iii) σ *and* σ' *are equivalent for* φ *w.r.t.* $r(i)$ *and* $r'(i)$*.*

(iv) for every $i \geq 0$*, if* $r(i) \to r(i+1)$ *then* $r'(i) \to r'(i+1)$*, and if* $r(i) \sim_j r(i+1)$*, for some* j*, then* $r'(i) \sim_j r'(i+1)$*.*

Proof. Let r be a t.e. run satisfying the lemma's hypothesis. We inductively build r' and show that the conditions above are satisfied. For $i = 0$, let $r'(0) = s'$. By hypothesis, r is s.t. $|U'| \geq |adom(r(0)) \cup adom(r(1)) \cup Con \cup \sigma(free(\varphi))|$. Thus, since $r(0) \rightsquigarrow r(1)$, by Lemma 3.9 there exists $t' \in \mathcal{S}'$ s.t. $r'(0) \rightsquigarrow t'$, $r(1) \approx t'$, and σ and σ' are equivalent for φ w.r.t. $r(1)$ and t'. Let $r'(1) = t'$. Lemma 3.9 guarantees that the transitions $r'(0) \rightsquigarrow t'$ and $r(0) \rightsquigarrow r(1)$ can be chosen so that they are either both temporal or both epistemic with the same index.

The case for $i > 0$ is similar. Assume that $r(i) \approx r'(i)$ and σ and σ' are equivalent for φ w.r.t. $r(i)$ and $r'(i)$. Since $r(i) \rightsquigarrow r(i+1)$ and $|U'| \geq |adom(r(i)) \cup adom(r(i+1)) \cup Con \cup \sigma(free(\varphi))|$, by Lemma 3.9 there exists $t' \in \mathcal{S}'$ s.t. $r'(i) \rightsquigarrow t'$, σ and σ' are equivalent for φ w.r.t. $r(i+1)$ and t', and $r(i+1) \approx t'$. Let $r'(i+1) = t'$. It is clear that r' is a t.e. run in $\mathcal{P}'$, and that, by Lemma 3.9, the transitions of r' can be chosen so as to fulfil requirement (iv). $\square$

We can now prove that FO-CTLK formulas cannot distinguish bisimilar and uniform AC-MAS. This is in marked contrast with the earlier example in this section which related to bisimilar but non-uniform AC-MAS.

Theorem 3.11 *Consider two bisimilar and uniform AC-MAS* $\mathcal{P}$ *and* $\mathcal{P}'$*, two bisimilar states* $s \in \mathcal{S}$ *and* $s' \in \mathcal{S}'$*, an FO-CTLK formula* φ*, and two assignments* σ *and* σ' *equivalent for* φ *w.r.t.* s *and* s'*. If*

1. *for every t.e. run* r *s.t.* $r(0) = s$*, for all* $k \geq 0$ *we have* $|U'| \geq |adom(r(k)) \cup adom(r(k+1)) \cup Con \cup \sigma(free(\varphi))| + |var(\varphi) \setminus free(\varphi)|$*; and*
2. *for every t.e. run* r' *s.t.* $r'(0) = s'$*, for all* $k \geq 0$ *we have* $|U| \geq |adom(r'(k)) \cup adom(r'(k+1)) \cup Con \cup \sigma'(free(\varphi))| + |var(\varphi) \setminus free(\varphi)|$*;*

then

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

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 from Prop. 3.3. The inductive cases for propositional connectives are straightforward.

For $\varphi \equiv \forall x \psi$, assume that $x \in free(\psi)$ (otherwise consider ψ, and the corresponding case), and no variable is quantified more than once (otherwise rename variables). 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 $\sigma\binom{x}{u}$. By definition, $\gamma(u) \in adom(s')$, and $\sigma'\binom{x}{\gamma(u)}$ is well-defined. Note that $free(\psi) = free(\varphi) \cup \{x\}$; so $\sigma\binom{x}{u}$ and $\sigma'\binom{x}{\gamma(u)}$ are equivalent for ψ w.r.t. s and s'. Moreover, $|\sigma\binom{x}{u}(free(\psi))| \leq |\sigma(free(\varphi))| + 1$, as u may not occur in $\sigma(free(\varphi))$. The same considerations apply to σ'. Further, $|var(\psi) \setminus free(\psi)| = |var(\varphi) \setminus free(\varphi)| - 1$, as $var(\psi) = var(\varphi)$,

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

For $\varphi \equiv AX\psi$, assume by contradiction that $(\mathcal{P}, s, \sigma) \models \varphi$ but $(\mathcal{P}', s', \sigma') \not\models \varphi$. Then, there exists a run r' s.t. $r'(0) = s'$ and $(\mathcal{P}', r'(1), \sigma') \not\models \psi$. Since $|\mathit{var}(\varphi) \setminus \mathit{free}(\varphi)| \geq 0$, by Lemma 3.10, there exists a run r s.t. $r(0) = s$, and for all $i \geq 0$, $r(i) \approx r'(i)$ and σ and σ' are equivalent for ψ w.r.t. $r(i)$ and $r'(i)$. Since r is a run s.t. $r(0) = s$, it satisfies hypothesis (1). Moreover, the same hypothesis is necessarily satisfied by all the t.e. runs r'' s.t. for some $i \geq 0$, $r''(0) = r(i)$ (otherwise, the t.e. run $r(0) \rightsquigarrow \cdots \rightsquigarrow r(i) \rightsquigarrow r''(1) \rightsquigarrow r''(2) \rightsquigarrow \cdots$ would not satisfy the hypothesis for r); the same considerations apply w.r.t hypothesis (2) and for all the t.e. runs r''' s.t. $r'''(0) = r'(i)$, for some $i \geq 0$. In particular, these hold for $i = 1$. Thus, we can inductively apply the lemma, by replacing s with $r(1)$, s' with $r'(1)$, and φ with ψ (observe that $\mathit{var}(\varphi) = \mathit{var}(\psi)$ and $\mathit{free}(\varphi) = \mathit{free}(\psi)$). But then we obtain $(\mathcal{P}, r(1), \sigma) \not\models \psi$, thus $(\mathcal{P}, r(0), \sigma) \not\models AX\psi$. This is a contradiction.

For $\varphi \equiv E\psi U\phi$, assume that the only variables common to ψ and ϕ occur free in both formulas (otherwise rename the quantified variables). Let r be a run s.t. $r(0) = s$, and there exists $k \geq 0$ s.t. $(\mathcal{P}, r(k), \sigma) \models \phi$, and $(\mathcal{P}, r(j), \sigma) \models \psi$ for $0 \leq j < k$. By Lemma 3.10 there exists a run r' s.t. $r'(0) = s'$ and for all $i \geq 0$, $r'(i) \approx r(i)$ 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 $\gamma_{i,\psi} = \gamma_i|_{\mathit{adom}(r(i)) \cup \mathit{Con} \cup \sigma(\mathit{free}(\psi))}$ and $\gamma_{i,\phi} = \gamma_i|_{\mathit{adom}(r(i)) \cup \mathit{Con} \cup \sigma(\mathit{free}(\phi))}$. Since $\mathit{free}(\psi) \subseteq \mathit{free}(\varphi)$,$\mathit{free}(\phi) \subseteq \mathit{free}(\varphi)$, 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 t.e. runs r'' s.t. $r''(0) = r(i)$, for some $i \geq 0$, and hypothesis (2) holds for all the t.e. runs r''' s.t. $r'''(0) = r'(i)$. Now observe that $|\sigma(\mathit{free}(\phi))|, |\sigma(\mathit{free}(\psi))| \leq |\sigma(\mathit{free}(\varphi))|$. Moreover, by the assumption on the common variables of ψ and ϕ, $(\mathit{var}(\varphi) \setminus \mathit{free}(\varphi)) = (\mathit{var}(\psi) \setminus \mathit{free}(\psi)) \uplus (\mathit{var}(\phi) \setminus \mathit{free}(\phi))$, thus $|\mathit{var}(\varphi) \setminus \mathit{free}(\varphi)| = |(\mathit{var}(\psi) \setminus \mathit{free}(\psi)| + |(\mathit{var}(\phi) \setminus \mathit{free}(\phi)|$, hence $|(\mathit{var}(\psi) \setminus \mathit{free}(\psi)|, |(\mathit{var}(\phi) \setminus \mathit{free}(\phi)| \leq |\mathit{var}(\varphi) \setminus \mathit{free}(\varphi)|$. 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 s.t. $r'(0) = s'$, $(\mathcal{P}', r'(k), \sigma') \models \phi$, 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 contradiction that $(\mathcal{P}, s, \sigma) \models \varphi$ but $(\mathcal{P}', s', \sigma') \not\models \varphi$. Then, there exists a run r' s.t. $r'(0) = s'$ and for every $k \geq 0$, either $(\mathcal{P}', r'(k), \sigma') \not\models \phi$ or there exists j s.t. $0 \leq j < k$ and $(\mathcal{P}', r'(j), \sigma') \not\models \psi$. By Lemma 3.10 there exists a run r s.t. $r(0) = s$, and for all $i \geq 0$, $r(i) \approx r'(i)$ 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 s.t. $r(0) = s$ and for every $k \geq 0$, either $(\mathcal{P}, r(k), \sigma) \not\models \phi$ or there exists j s.t. $0 \leq j < k$ and $(\mathcal{P}, r(j), \sigma) \not\models \psi$, that is, $(\mathcal{P}, s, \sigma) \not\models A\psi U\phi$. This is a contradiction.

For $\varphi \equiv K_i\psi$, assume by contradiction that $(\mathcal{P}, s, \sigma) \models \varphi$ but $(\mathcal{P}', s', \sigma') \not\models \varphi$. Then, there exists s'' s.t. $s' \sim_i s''$ and $(\mathcal{P}', s'', \sigma') \not\models \psi$. By Lemma 3.10 there exists s''' s.t. $s''' \approx s''$, $s \sim_i s'''$, and σ and σ' are equivalent for ψ w.r.t. s'' and s'''. Thus, by an argument analogous to that used for the case of AX, we can apply the induction hypothesis, obtaining $(\mathcal{P}, s''', \sigma) \not\models \psi$. But then $(\mathcal{P}, s, \sigma) \not\models K_i\psi$, which is a contradiction.

Finally, for $\varphi \equiv C\psi$, assume by contradiction that $(\mathcal{P}, s, \sigma) \models \varphi$ but $(\mathcal{P}', s', \sigma') \not\models \varphi$. Then, there exists an s'' s.t. $s' \sim s''$ and $(\mathcal{P}', s'', \sigma') \not\models \psi$. Again by Lemma 3.10 there exists s''' s.t. $s''' \approx s''$, $s \sim s'''$, and σ and σ' are equivalent for ψ w.r.t. s'' and s'''. Thus, by an argument analogous to that used for the case of K_i, we can apply the induction hypothesis, obtaining $(\mathcal{P}, s''', \sigma) \not\models \psi$. But then $(\mathcal{P}, s, \sigma) \not\models C\psi$, which is a contradiction. □

We can now easily extend the above result to the model checking problem for AC-MAS.

Theorem 3.12 *Consider two bisimilar and uniform AC-MAS $\mathcal{P}$ and $\mathcal{P}'$, and an FO-CTLK formula φ.*

If

1. *for all t.e. runs r s.t. $r(0) = s_0$, and for all $k \geq 0$, $|U'| \geq |adom(r(k)) \cup adom(r(k+1)) \cup Con| + |var(\varphi)|$, and*
2. *for all t.e. runs r' s.t. $r'(0) = s'_0$, and for all $k \geq 0$, $|U| \geq |adom(r'(k)) \cup adom(r'(k+1)) \cup Con| + |var(\varphi)|$*

then

$$\mathcal{P} \models \varphi \quad \textit{iff} \quad \mathcal{P}' \models \varphi.$$

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 3.11. Further, notice that, by cardinality considerations, given the assignment $\sigma : Var \mapsto U$, there exists an assignment $\sigma' : Var \mapsto U'$ s.t. σ and σ' are equivalent for φ w.r.t. s_0 and s'_0. Thus, by applying Theorem 3.11 we have that if there exists an assignment σ s.t. $(\mathcal{P}, s_0, \sigma) \not\models \varphi$, then there exists an assignment σ' s.t. $(\mathcal{P}', s'_0, \sigma') \not\models \varphi$. The converse can be proved analogously, as the hypotheses are symmetric. □

This result shows that uniform AC-MAS can in principle be verified by model checking a bisimilar one. Note that this applies to an infinite AC-MAS $\mathcal{P}$, as well. In this case the results above enable us to show that the verification question can be posed on the corresponding, possibly finite $\mathcal{P}'$ as long as U', as defined above, is sufficiently large for $\mathcal{P}'$ to bisimulate $\mathcal{P}$. A noteworthy class of infinite systems for which these results prove particularly powerful is that of bounded AC-MAS, which, as discussed in the next subsection, always admit a finite abstraction.

3.3 Finite Abstractions

We now define a notion of finite abstraction for AC-MAS, and prove that, under uniformity, abstractions are bisimilar to the corresponding concrete model. We are particularly interested in finite abstractions; so we operate on a special class of infinite models that we call *bounded*.

Definition 3.13 (Bounded AC-MAS) *An AC-MAS $\mathcal{P}$ is b-*bounded*, for $b \in \mathbb{N}$, if for all $s \in \mathcal{S}$, $|adom(s)| \leq b$.*

An AC-MAS is b-bounded if none of its reachable states contain more than b distinct elements. Observe that bounded AC-MAS may be defined on infinite domains. Furthermore, note that a b-bounded AC-MAS may contain infinitely many states, all bounded by b. So b-bounded systems are infinite-state in general. Notice also that the value b constrains only the number of distinct individuals in a state, not the *size* of the state itself, intended as the amount of memory required to accommodate the individuals. Indeed, the infinitely many elements in a domain U need an unbounded number of bits to be represented (e.g., as finite strings), so, even though each state is guaranteed to contain at most b distinct elements, nothing can be said about how large the actual space required by such elements is. Conversely, memory-bounded AC-MAS are finite-state (hence b-bounded, for some b).

Since b-bounded AC-MAS are in general memory-unbounded, they cannot be verified by trivially generating and checking all their possibly infinite executions. However, we will show later that any b-bounded and uniform infinite-state AC-MAS admits a finite-state abstraction which can be used to verify it.

We introduce abstractions in a modular manner by first introducing a set of abstract agents from a concrete AC-MAS.

Definition 3.14 (Abstract agent) *Let $A = \langle \mathcal{D}, Act, Pr \rangle$ be an agent defined on the interpretation domain U. Given an interpretation domain U', the* abstract agent *of A on U' is the agent $A' = \langle \mathcal{D}', Act', Pr' \rangle$ such that:*

1. $\mathcal{D}' = \mathcal{D}$;
2. $Act' = Act$;
3. $\alpha(\vec{u}') \in Pr'(l')$, *with* $l' \in \mathcal{D}'(U')$, *iff there exist* $l \in \mathcal{D}(U)$ *and* $\alpha(\vec{u}) \in Pr(l)$ *s.t.* $l' \simeq l$, *for some witness* ι, *and* $\vec{u}' = \iota'(\vec{u})$, *for some bijection* ι' *extending* ι *to* $\vec{u}$.

Given a set Ag of agents defined on U, Ag' denotes the set of corresponding abstractions on U' of the agents in Ag.

We remark that the abstract agent A' is an agent in line with Definition 2.6. Notice that the protocol of A' is defined on the basis of its corresponding concrete agent A and requires the existence of a bijection between the elements in the corresponding local states and the action parameters. Thus, in order for a ground action of A to have a counterpart in A', the last requirement of Definition 3.14 constrains U' to contain a sufficient number of distinct values. As it will become apparent later, the size of U' determines how closely an abstract system can simulate its concrete counterpart. Notice also that, in general, an agent may not be an abstraction of itself on U, as for instance data may impact the agent's protocol.

Next, we combine the notion of uniformity with that of boundedness. Our aim is to identify conditions under which the verification of an infinite AC-MAS can be reduced to the verification of a finite one. The main result here is given by Corollary 3.19 which guarantees that, in the context of bounded AC-MAS, uniformity is a sufficient condition for bisimilar finite abstractions to be satisfaction-preserving.

In the following we assume that any AC-MAS $\mathcal{P}$ is such that *adom*$(s_0) \subseteq$ *Con*. If this is not the case, *Con* can be extended so as to include all the (finitely many) elements in *adom*(s_0). We start by formalising the notion of abstraction.

Definition 3.15 (Abstract AC-MAS) *Let $\mathcal{P} = \langle Ag, s_0, \tau \rangle$ be an AC-MAS and Ag' the set of abstract agents obtained as in Definition 3.14 for some interpretation domain U'. The AC-MAS $\mathcal{P}' = \langle Ag', s'_0, \tau' \rangle$ is said to be an* abstraction *of $\mathcal{P}$ iff:*

- $s'_0 = s_0$;
- $t' \in \tau'(s', \vec{\alpha}(\vec{u}'))$ *iff there exist $s, t \in \mathcal{S}$ and $\vec{\alpha}(\vec{u}) \in Act(U)$ such that $s \oplus t \simeq s' \oplus t'$, for some witness ι, $t \in \tau(s, \vec{\alpha}(\vec{u}))$, and $\vec{u}' = \iota'(\vec{u})$ for some bijection ι' extending ι to $\vec{u}$.*

It can be checked that $\mathcal{P}'$, as defined above, is indeed an AC-MAS as it satisfies the relevant conditions on protocols and transitions in Definition 2.7. Indeed, if $t' \in \tau'(s', \vec{\alpha}(\vec{u}'))$, then there exist $s, t \in \mathcal{S}$, and $\vec{\alpha}(\vec{u})$ such that $t \in \tau(s, \vec{\alpha}(\vec{u}))$, $s \oplus t \simeq s' \oplus t'$ for some witness ι, and $\vec{u} = \iota'(\vec{u}')$ for some bijection ι' extending ι. This means that $\alpha_i(\vec{u}_i) \in Pr_i(l_i)$ for $i \leq n$. By definition of Pr'_i we have that $\alpha_i(\vec{u}'_i) \in Pr'_i(l'_i)$ for $i \leq n$.

The definition requires abstractions to have initial states isomorphic to their concrete counterparts; specifically they have to be equal as $adom(s_0) \subseteq Con$. Moreover, the second constraint entails that a transition in the concrete model exists if and only if the same transition, up to renaming of the involved values, exists in the abstraction. So, for example, a copy action in the concrete model has a corresponding copy action in the abstract model. Crucially, this condition requires that the domain U' contains enough elements to bisimulate the concrete states and action effects. This will be made precise in Lemma 3.17.

Obviously, if U' has finitely many elements, then $\mathcal{S}'$ has finitely many states. Observe also that by varying U' we obtain different abstractions. Finally, notice that an AC-MAS is not necessarily an abstraction of itself. This issue is addressed in Lemma 3.16.

Next, we investigate the relationship between an AC-MAS and its abstractions. A first useful result states that every finite abstraction is uniform, independently of the properties of the AC-MAS they abstract.

Lemma 3.16 *Every abstraction $\mathcal{P}'$ of an AC-MAS $\mathcal{P}$ is uniform. Moreover, if $\mathcal{P}$ is uniform and $U' = U$, then $\mathcal{P}' = \mathcal{P}$.*

Proof. Consider $s, t, s' \in \mathcal{S}'$, $t' \in \mathcal{D}(U')$, and $\vec{\alpha}(\vec{u}) \in Act'(U')$ s.t. $t \in \tau'(s, \vec{\alpha}(\vec{u}))$ and $s \oplus t \simeq s' \oplus t'$, for some witness ζ. We need to show that $\mathcal{P}'$ admits a transition from s' to t'. Since $\mathcal{P}'$ is an abstraction of $\mathcal{P}$, given the definition of τ', there exist $s'', t'' \in \mathcal{S}$ and $\vec{\alpha}(\vec{u}'') \in Act(U)$ s.t. $t'' \in \tau(s'', \vec{\alpha}(\vec{u}''))$, $s'' \oplus t'' \simeq s \oplus t$, for some witness ι, and $\vec{u} = \iota'(\vec{u}'')$, for some constant-preserving bijection ι' extending ι to $\vec{u}''$. Consider $\vec{u}' \in U'^{|\vec{u}|}$ such that $\vec{u}' = \zeta'(\vec{u})$, for some constant-preserving bijection ζ' extending ζ to $\vec{u}$. Obviously, the composition $\zeta' \circ \iota'$ is a constant-preserving bijection such that $\vec{u}' = \zeta'(\iota'(\vec{u}''))$. Moreover, it can be restricted to a witness for $s'' \oplus t'' \simeq s' \oplus t'$. But then, since $\mathcal{P}'$ is an abstraction of $\mathcal{P}$, this implies that $t' \in \tau'(s', \vec{\alpha}(\vec{u}'))$. Thus, $\mathcal{P}'$ is uniform.

Moreover, to prove that $\mathcal{P}$ is an abstraction of itself every time $\mathcal{P}$ is uniform and $U' = U$, we notice that if the transition $t \in \tau(s, \vec{\alpha}(\vec{u}))$ is in $\mathcal{P}$, then it is also in $\mathcal{P}'$ by the definition of abstraction. Also, if the transition $t' \in \tau'(s', \vec{\alpha}(\vec{u}'))$ appears in $\mathcal{P}'$, then there exist $s, t \in \mathcal{S}$ and $\vec{\alpha}(\vec{u}) \in Act(\vec{U})$ s.t. $s \oplus t \simeq s' \oplus t'$ for some witness ι, $t \in \tau(s, \vec{\alpha}(\vec{u}))$, and $\vec{u}' = \iota'(\vec{u})$ for some constant-preserving bijection ι' extending ι to $\vec{u}$. Finally, since $\mathcal{P}$ is uniform it is the case that the transition $t' \in \tau'(s', \alpha(\vec{u}'))$ is in $\mathcal{P}$ as well. □

This lemma provides sufficient conditions under which an AC-MAS is an abstraction of itself, namely being uniform and having the same interpretation domain.

The second result below guarantees that every uniform, b-bounded AC-MAS is bisimilar to any of its abstractions, provided these are built over a sufficiently large interpretation domain. In the following, we take $N_{Ag} = N_{Ag'} = \sum_{A_i \in Ag} \max_{\alpha(\vec{x}) \in Act_i}\{|\vec{x}|\}$, i.e., N_{Ag} is the sum of the maximum number of parameters contained in the action types of each agent in Ag.

Lemma 3.17 *Consider a uniform, b-bounded AC-MAS $\mathcal{P}$ over an infinite interpretation domain U, and an interpretation domain U' such that $Con \subseteq U'$. If $|U'| \geq 2b + |Con| + N_{Ag}$, then any abstraction $\mathcal{P}'$ of $\mathcal{P}$ over U' is bisimilar to $\mathcal{P}$.*

Proof. Let $B = \{\langle s, s'\rangle \in \mathcal{S} \times \mathcal{S}' \mid s \simeq s'\}$. We prove that B is a bisimulation such that $\langle s_0, s_0'\rangle \in B$. We start by proving that B is a simulation relation. To this end, observe that since $s_0 = s_0'$, then $s_0 \simeq s_0'$, and $\langle s_0, s_0'\rangle \in B$. Next, consider $\langle s, s'\rangle \in B$, thus $s \simeq s'$. Assume that $s \to t$, for some $t \in \mathcal{S}$. Then, there must exist $\vec{\alpha}(\vec{u}) \in Act(U)$ such that $t \in \tau(s, \vec{\alpha}(\vec{u}))$. Moreover, since $|U'| \geq 2b + |Con| + N_{Ag}$, $\sum_{A_i \in Ag} |\vec{u}_i| \leq N_{Ag}$, and $|adom(s) \cup adom(t)| \leq 2b$, the witness ι for $s \simeq s'$ can be extended to $\bigcup_{A_i \in Ag} \vec{u}_i$ as a bijection ι'. Now let $t' = \iota'(t)$. By the way ι' has been defined, it can be seen that $s \oplus t \simeq s' \oplus t'$. Further, since $\mathcal{P}'$ is an abstraction of $\mathcal{P}$, we have that $t' \in \tau'(s', \vec{\alpha}(\vec{u}'))$ for $\vec{u}' = \iota'(\vec{u})$, that is, $s' \to t'$ in $\mathcal{P}'$. Therefore, there exists $t' \in \mathcal{S}'$ such that $s' \to t'$, $s \oplus t \simeq s' \oplus t'$, and $\langle t, t'\rangle \in B$. As regards the epistemic relation, assume $s \sim_i t$ for some $i \in \{1, \ldots, n\}$ and $t \in \mathcal{S}$. By definition of $\sim_i$, $l_i(s) = l_i(t)$. Since $|U'| \geq 2b + |Con|$, any witness ι for $s \simeq s'$ can be extended to a witness ι' for $s \oplus t \simeq s' \oplus t'$, where $t' = \iota'(t)$. Obviously, $l_i(s') = l_i(t')$. Thus, to prove that $s' \sim_i t'$, we are left to show that $t' \in \mathcal{S}'$, i.e., that t' is reachable in $\mathcal{P}'$ from $s_0' = s_0$. To this end, observe that since $t \in \mathcal{S}$, there exists a purely temporal run r such that $r(0) = s_0$ and $r(k) = t$, for some $k \geq 0$. Thus, there exist $\vec{\alpha}^1(\vec{u}^1) \ldots, \vec{\alpha}^k(\vec{u}^k)$ such that $r(j+1) \in \tau(r(j), \vec{\alpha}^{j+1}(\vec{u}^{j+1}))$, for $0 \leq j < k$. Since $|U'| \geq 2b + |Con|$, we can define, for $0 \leq j < k$, a function ι_j that is a witness for $r(j) \oplus r(j+1) \simeq \iota_j(r(j)) \oplus \iota_j(r(j+1))$. In particular, this can be done starting from $j = k-1$, defining ι_{k-1} so that $\iota_{k-1}(r(k)) = \iota_{k-1}(t) = t'$, and proceeding backward to $j = 0$, so that, for $0 \leq j < k$, we have $\iota_j(r(j+1)) = \iota_{j+1}(r(j+1))$. Observe that since $adom(s_0) \subseteq Con$, necessarily $i_0(r(0)) = i_0(s_0) = s_0 = s_0'$. Moreover, as $|U'| \geq 2b + |Con| + N_{Ag}$, each ι_j can be extended to a bijection ι_j', to the elements occurring in $\vec{u}^{j+1}$. Thus, given that $\mathcal{P}'$ is an abstraction of $\mathcal{P}$, for $0 \leq j < k$, we have that $\iota_j'(r(j+1)) \in \tau(\iota_j'(r(j)), \vec{\alpha}(\iota_j'(\vec{u}^{j+1})))$. Hence, the sequence $\iota_0'(r(0)) \to \cdots \to \iota_{k-1}'(r(k))$ is a run of $\mathcal{P}'$, and, since $t' = \iota_{k-1}'(r(k))$, t' is reachable in $\mathcal{P}'$. Therefore $s' \sim_i t'$. Further, since $t \simeq t'$, by definition of B, it is the case that $\langle t, t'\rangle \in B$, hence B is a simulation.

To prove that B^{-1} is a simulation, given $\langle s, s'\rangle \in B$ (thus $s \simeq s'$), assume that $s' \to t'$, for some $t' \in \mathcal{S}'$. Obviously, there exists $\vec{\alpha}(\vec{u}') \in Act(U')$ such that $t' \in \tau'(s', \vec{\alpha}(\vec{u}'))$. Because $\mathcal{P}'$ is an abstraction of $\mathcal{P}$, there exist $s'', t'' \in \mathcal{S}$ and $\vec{\alpha}(\vec{u}'') \in Act(U)$ such that $s'' \oplus t'' \simeq s' \oplus t'$, for some witness ι, and $t'' \in \tau(s'', \alpha(\vec{u}''))$, with $\vec{u}'' = \iota'(\vec{u}')$, for some bijection ι' extending ι to $\vec{u}'$. Observe that $s' \simeq s''$, thus, by transitivity of $\simeq$ we have $s \simeq s''$. The fact that there exists $t \in \mathcal{S}$ such that $s \to t$ easily follows from the uniformity of $\mathcal{P}$. Thus, since $t' \simeq t$, we have $\langle t, t'\rangle \in B$. For the epistemic relation, assume $s' \sim_i t'$ for some $t' \in \mathcal{S}'$ and $0 < i \leq n$. Let ι be a witness for $s' \simeq s$, and let ι' be an extension of ι that is a witness for $s' \oplus t' \simeq s \oplus t$. For $t = \iota'(t')$, it can be seen that $l_i(s) = l_i(t)$. Observe that $t' \in \mathcal{S}'$. Using an argument analogous to the one above, but exploiting the fact that $\mathcal{P}$ is uniform, that $\mathcal{P}'$ is certainly b-bounded, and that $|U| > 2b + |Con| + N_{Ag}$ as U is infinite, we can show that $t \in \mathcal{S}$ by constructing a run r of $\mathcal{P}$ such that $r(k) = t$, for some $k \geq 0$.

Then $s \sim_i t$. Further, since $t' \simeq t$, we have $\langle t, t' \rangle \in B$. Therefore, B^{-1} is a simulation. So, $\mathcal{P}$ and $\mathcal{P}'$ are bisimilar. □

This result allows us to prove our main abstraction theorem.

Theorem 3.18 *Consider a b-bounded and uniform AC-MAS $\mathcal{P}$ over an infinite interpretation domain U, an FO-CTLK formula φ, and an interpretation domain U' such that Con $\subseteq U'$. If $|U'| \geq 2b + |Con| + \max\{|vars(\varphi)|, N_{Ag}\}$, then for any abstraction $\mathcal{P}'$ of $\mathcal{P}$ over U', we have that:*

$$\mathcal{P} \models \varphi \quad \textit{iff} \quad \mathcal{P}' \models \varphi.$$

Proof. By Lemma 3.16, $\mathcal{P}'$ is uniform. Thus, by the hypothesis on the cardinalities of U and U', Lemma 3.17 applies, so $\mathcal{P}$ and $\mathcal{P}'$ are bisimilar. Obviously, also $\mathcal{P}'$ is b-bounded. Thus, since $\mathcal{P}$ and $\mathcal{P}'$ are b-bounded, and by the cardinality hypothesis on U and U', Theorem 3.12 applies. In particular, notice that for every temporal-epistemic run r s.t. $r(0) = s_0$, and for all $k \geq 0$, we have that $|U'| \geq |adom(r(k)) \cup adom(r(k+1)) \cup Con| + |var(\varphi)|$, as $|adom(r(k))| \leq b$ by b-boundedness. Therefore, $\mathcal{P} \models \varphi$ iff $\mathcal{P}' \models \varphi$. □

It follows that by using a sufficiently large number of abstract values in U', we can reduce the verification of an infinite, bounded, and uniform AC-MAS to the verification of a finite one.

Corollary 3.19 *Given a b-bounded and uniform AC-MAS $\mathcal{P}$ over an infinite interpretation domain U, and an FO-CTLK formula φ, there exists an AC-MAS $\mathcal{P}'$ over a finite interpretation domain U' such that $\mathcal{P} \models \varphi$ iff $\mathcal{P}' \models \varphi$.*

It should also be noted that U' can simply be taken to be any finite subset of U (including *Con*) satisfying the cardinality requirement above. By doing so, the finite abstraction $\mathcal{P}'$ can be defined simply as the restriction of $\mathcal{P}$ to U'. Thus, every infinite, b-bounded and uniform AC-MAS is bisimilar to a finite subsystem, which then satisfies the same formulas.

Note that we are not concerned with the actual construction of the finite abstraction. This is because we intend to construct it directly from an artifact-centric program, as we will do in Section 4. Before that, we explore the complexity of the model checking problem.

3.4 The Complexity of Model Checking Finite AC-MAS against FO-CTLK Specifications

We now analyse the complexity of the model checking problem for finite AC-MAS with respect to FO-CTLK specifications. The input of the problem consists of an AC-MAS $\mathcal{P}$ on a finite domain U and an FO-CTLK formula φ; the output is an assignment σ such that $(\mathcal{P}, s_0, \sigma) \models \varphi$, whenever the property is satisfied. Hereafter we follow standard literature for basic notions and definitions (Grohe, 2001).

To encode an AC-MAS $\mathcal{P}$ we use a tuple $E_{\mathcal{P}} = \langle U, \mathcal{D}, s_0, \Phi_\tau \rangle$, where U is the (finite) interpretation domain, $\mathcal{D}$ is the global database schema, s_0 is the initial state, and $\Phi_\tau = \{\phi_{\vec{\alpha}_1}, \ldots, \phi_{\vec{\alpha}_m}\}$ is a set of FO-formulas, each capturing the transitions associated with a ground joint action $\vec{\alpha}_i$. Since U is finite, so is the set of ground actions, thus Φ_τ. Each $\varphi_{\vec{\alpha}_i}$ is a FO-formula over the alphabet $\mathcal{D}_{Ag} \cup \mathcal{D}'_{Ag}$, where $\mathcal{D}_{Ag} = \{P_i^j/q_i \mid P_i/q_i \in \mathcal{D}, j \leq n\}$ is the set containing one distinct relation symbol P_i^j, for each agent $j \leq n$ and the relation symbol $P_i \in \mathcal{D}$. We take Φ_τ such that $s' \in \tau(s, \vec{\alpha})$

iff $D_{Ag} \oplus D'_{Ag} \models \phi_{\vec{\alpha}}$, for $s, s' \in \mathcal{D}(U)$, such that for every $P_i \in \mathcal{D}$ and $j \leq n$, $l_j(P_i) = D_{Ag}(P_i^j)$ and $l'_j(P_i) = D'_{Ag}(P_i^j)$.

As an example, for $\mathcal{D} = \{P\}$ (thus $\mathcal{D}_{Ag} = \{P^j \mid j \leq n\}$) and an action type α with no parameters, consider the formula $\phi_{\vec{\alpha}} = \bigwedge_{j=0}^{n} \forall x P^{j'}(x) \leftrightarrow \neg P^j(x)$, which intuitively captures all transitions in which in the successor state predicate P contains all and only those elements of U that in the current state are not in P.

It can be proved that every transition relation τ can be represented as discussed above, and that, given $E_{\mathcal{P}}$, the size $||\mathcal{P}|| \doteq |\mathcal{S}| + |\tau|$ of the encoded AC-MAS $\mathcal{P}$ is such that $||\mathcal{P}|| \leq |Act| \cdot |U|^{p_{max}} \cdot 2^{3\ell q_{max}}$, where: p_{max} is the largest number of parameters in some action type of Act, ℓ is the number of relation symbols in $\mathcal{D}$, and q_{max} is the largest arity of such symbols. This corresponds to a doubly exponential bound for $||\mathcal{P}||$ w.r.t. $||E_{\mathcal{P}}|| \doteq |U| + ||\mathcal{D}|| + |\Phi_\tau|$, where $||\mathcal{D}|| = \sum_{P_k \in \mathcal{D}} q_k$, for q_k the arity of P_k. Specifically, we have $||\mathcal{P}|| \leq 2^{3 \cdot 2^{||E_{\mathcal{P}}||^4}}$.

We carry out the complexity analysis on the basis of the input above; clearly the same results apply for equally compact inputs such as the AC programs to be presented in Section 4.

We consider the *combined complexity* of the input, that is, $||E_{\mathcal{P}}|| + ||\varphi||$. We say that the combined complexity of model checking finite AC-MAS against FO-CTLK specifications is EXPSPACE-complete if the problem is in EXPSPACE, i.e., there is a polynomial $p(x)$ and an algorithm solving the problem in space bounded by $2^{p(||E_{\mathcal{P}}||+||\varphi||)}$, and the problem is EXPSPACE-hard, i.e., every EXPSPACE problem can be reduced to model checking finite AC-MAS against FO-CTLK specifications.

Theorem 3.20 *The model checking problem for finite AC-MAS succinctly presented as above against FO-CTLK specifications is EXPSPACE-complete.*

Proof. To show that the problem is in EXPSPACE, recall that $||\mathcal{P}||$ is at most doubly exponential w.r.t. the size of the input, thus so is $|\mathcal{S}|$. We describe an algorithm that works in NEXPSPACE; this combines the algorithm for model checking the first-order fragment of FO-CTLK and that for the temporal-epistemic fragment. Since NEXPSPACE = EXPSPACE, the result follows. Given an AC-MAS $\mathcal{P}$ and an FO-CTLK formula φ, we guess an assignment σ and check whether $(\mathcal{P}, s_0, \sigma) \models \varphi$. This can be done by induction according to the structure of φ. If φ is atomic, this check can be done in polynomial time w.r.t. the size of the state it is evaluated on, that is, exponential time w.r.t. $||E_{\mathcal{P}}||$. If φ is of the form $\forall x \psi$, then we can apply the algorithm for model checking first-order (non-modal) logic, which works in PSPACE. Finally, if the outmost operator in φ is either a temporal or epistemic modality, then we can extend the automata-based algorithm to model check propositional CTL (Kupferman, Vardi, & Wolper, 2000; Lomuscio & Raimondi, 2006), which works in logarithmic space in $|\mathcal{S}|$. However, we remarked above that $|\mathcal{S}|$ is generally doubly exponential in $||E_{\mathcal{P}}||$. Thus, this step can be performed in space singly exponential in $||E_{\mathcal{P}}||$. All these steps can be performed in time polynomial in the size of φ. As a result, the total combined complexity of model checking finite AC-MAS is in NEXPSPACE = EXPSPACE.

To prove that the problem is EXPSPACE-hard we show a reduction from any problem in EXPSPACE. We assume standard definitions of Turing machines and reductions (Papadimitriou, 1994). If A is a problem in EXPSPACE, then there exists a deterministic Turing machine $\mathcal{T}_A = \langle \mathcal{Q}, \Sigma, q_0, \mathcal{F}, \delta \rangle$, where $\mathcal{Q}$ is the finite set of states, Σ the machine alphabet, $q_0 \in \mathcal{Q}$ the initial state, $\mathcal{F}$ the set of accepting states, and δ the transition function, that solves A using at most space $2^{p(|in|)}$ on a given input in, for some polynomial function p. As standard, we assume δ to be a relation on $(\mathcal{Q}\times\Sigma\times\mathcal{Q}\times\Sigma\times D)$, with $D=\{L,R\}$, and $\langle q,c,q',c',d\rangle\in\delta$ representing a transition from state q to state q', with characters c and c' read and written respectively , and head direction d ((L)eft and (R)ight). Without loss of generality, we assume that $\mathcal{T}_A$ uses only the righthand half of the tape.

From $\mathcal{T}_A$ and in, we build an encoding $E_{\mathcal{P}}=\langle\mathcal{D},U,s_0,\Phi_\tau\rangle$ of an AC-MAS $\mathcal{P}$ induced by a single (environment) agent $A_E=\langle\mathcal{D}_E,Act_E,Pr_E\rangle$ defined on $U=\Sigma\cup\mathcal{Q}\cup\{0,1\}$, where: (i) $\mathcal{D}_E=\{P/p(|in|)+1,Q/1,H/p(|in|),F/1\}$; (ii) Act_E is the singleton $\{\alpha_E\}$, with α_E parameter-free; (iii) $\alpha_E\in Pr_E(l_E)$ for every $l_E\in\mathcal{D}(U)$. Intuitively, the states of $\mathcal{P}$ correspond to configurations of $\mathcal{T}_A$, while τ mimics δ. To define $E_{\mathcal{P}}$, we let $\mathcal{D}=\mathcal{D}_E$. The intended meaning of the predicates in $\mathcal{D}$ is as follows: the first $p(|in|)$ elements of a P-tuple encode (in binaries) the position of a non-blank cell, and the $(p(|in|)+1)$-th element contains the symbol appearing in that cell; Q contains the current state q of $\mathcal{T}_A$; H contains the position of the cell the head is currently on; F contains the final states of $\mathcal{T}_A$, i.e., $F=\mathcal{F}$. The initial state s_0 represents the initial configuration of $\mathcal{T}_A$, that is, for $in=in_0\cdots in_\ell$: $s(Q)=\{q_0\}$; $s(H)=\{\langle 0,\ldots,0\rangle\}$; and $s(P)=\{\langle\textsc{bin}(i),in_i\rangle\mid i\in\{0,\ldots,\ell\}\}$, where $\textsc{bin}(i)$ stands for the binary encoding in $p(|in|)$ bits of the integer i. Observe that $p(|in|)$ bits are enough to index the (at most) $2^{p(|in|)}$ cells used by $\mathcal{T}_A$.

As to the transition relation, we define $\Phi_\tau=\{\phi_{\alpha_E}\}$, where (we avoid sub- and superscripts in predicate symbols, i.e., $\mathcal{D}=\mathcal{D}_{Ag}$ as no ambiguity can arise with only one agent):

$$
\begin{aligned}
\phi_{\alpha_E} = \bigvee_{\langle q,c,q',c',d\rangle\in\delta} & (\forall x F(x)\leftrightarrow F'(x))\wedge && (1)\\
& Q(q)\wedge(\forall x Q(x)\rightarrow x=q)\wedge Q'(q')\wedge(\forall x Q'(x)\rightarrow x=q')\wedge && (2)\\
& \exists\vec{p}(H(\vec{p})\wedge(\forall x H(x)\rightarrow x=\vec{p})\wedge(P(\vec{p},c)\vee(c=\Box\wedge\neg\exists x P(\vec{p},x))))\wedge && (3)\\
& \exists\vec{p'}(d=R\rightarrow\textsc{succ}(\vec{p},\vec{p'}))\wedge(d=L\rightarrow\textsc{succ}(\vec{p'},\vec{p}))\wedge H'(\vec{p'})\wedge(\forall x H'(x)\rightarrow x=\vec{p'})\wedge && (4)\\
& (P'(\vec{p},c')\leftrightarrow(c'\neq\Box))\wedge(\forall x P'(\vec{p},x)\rightarrow x=c')\wedge && (5)\\
& (\forall\vec{x},y(P(\vec{x},y)\wedge(\vec{x}\neq\vec{p})\rightarrow P'(\vec{x},y))\wedge(\forall\vec{x},y P'(\vec{x},y)\rightarrow(P(\vec{x},y)\vee(\vec{x}=\vec{p}\wedge y=c')))) && (6)
\end{aligned}
$$

The symbol $\Box$ represents the content of blank cells, while $\textsc{succ}(\vec{x},\vec{x'})=\bigwedge_{i=1}^{p(|in|)}(x'_i=0\vee x'_i=1)\wedge(x'_i=1\leftrightarrow((x'_i=0\wedge\bigwedge_{j=1}^{i-1}x_j=1)\vee(x'_i=1\wedge\neg\bigwedge_{j=1}^{i-1}x_j=1)))$ is a formula capturing that $\vec{x'}$ is the successor of $\vec{x}$, for $\vec{x}$ and $\vec{x'}$ interpreted as $p(|in|)$-bit binary encodings of integers (observe that $\{0,1\}\in U$). Such a formula can obviously be written in polynomial time w.r.t. $p(|in|)$, as well as $E_{\mathcal{P}}$, and in particular s_0 and ϕ_{α_E}. Formula ϕ_{α_E} is obtained as a disjunction of subformulas, each referring to a transition of δ. For each subformula, i.e., transition $\langle q,c,q',c',d\rangle$: line 1 expresses that F, which encodes the final states of the machine, does not change along the transition (this formula could be moved out of the big disjunction); line 2 encodes that the machine will be in exactly one state, q', after the transition takes place; line 3 expresses that the symbol read by the head is c (possibly blank); line 4 captures that the head moves in direction d; line 5 states that the head writes symbol c on the cell, before moving; finally, line 6 states that the content of the tape does not change, except for the cell that the head is on.

The obtained transition function is such that $\tau(s,\alpha_E)=s'$ iff, for $\delta(q,c)=(q',c',d)$ in $\mathcal{T}_A$, we have that: $s'(P)$ is obtained from $s(P)$ by overwriting with c' (if not blank) the symbol in position $(p(|in|)+1)$ of the tuple in $s(P)$ beginning with the $p(|in|)$-tuple $s(H)$ (that is, c by definition of ϕ_{α_E}); by updating $s(H)$ according to d, that is by increasing or decreasing the value it contains; and by setting $s'(Q) = \{q'\}$. The predicate F does not change. Observe that cells not occurring in P are interpreted as if containing $\Box$ and when $\Box$ is to be written on a cell, the cell is simply removed from P.

It can be checked that, starting with $s = s_0$, by iteratively generating the successor state s' according to Φ_τ, i.e., s' s.t. $s \oplus s' \models \phi_{\alpha_E}$, one obtains a (single) $\mathcal{P}$-run that is a representation of the computation of $\mathcal{T}_A$ on in, where each pair of consecutive $\mathcal{P}$-states corresponds to a computation step. In particular, at each state, Q contains the current state of $\mathcal{T}_A$. It should be clear that $\varphi = EF(\exists x Q(x) \wedge F(x))$ holds in $\mathcal{P}$ iff $\mathcal{T}_A$ accepts in. Thus, by model checking φ on $\mathcal{P}$, we can check whether $\mathcal{T}_A$ accepts in. This completes the proof of EXPSPACE-hardness. $\Box$

Note that the result above is given in terms of the "data structures" in the model, i.e., U and $\mathcal{D}$, and not the state space $\mathcal{S}$ itself. This accounts for the high complexity of model checking AC-MAS, as the state space is doubly exponential in the size of the data. By analysing the refined bound on the size of $||\mathcal{P}||$ ($||\mathcal{P}|| \leq |Act| \cdot |U|^{p_{max}} \cdot 2^{3\ell q_{max}}$), it can be seen that the double exponential is essentially due to the number of parameters in action types, the number of relation symbols occurring in $\mathcal{D}$, and their respective arities. Thus, for fixed database schema and set of action types, the resulting space complexity is reduced to singly exponential.

While EXPSPACE-hardness indicates intractability, we note that this is to be expected given that we are dealing with quantified structures which are in principle prone to high complexity. Recall also from Section 3.3 that the size of the interpretation domain U' of the abstraction $\mathcal{P}'$ is linear in the bound b, the number of constants in *Con*, the size of ϕ, and N_{Ag}. Hence, model checking bounded and uniform AC-MAS is EXPSPACE-complete with respect to these elements, whose size will generally be small. Thus, we believe that in several cases of practical interest model checking AC-MAS may be entirely feasible.

4. Artifact-Centric Programs

We have so far developed a formalism that can be used to specify and reason about temporal-epistemic properties of models representing artifact-centric systems. We have identified a notable class of models that admit finite abstractions. As we remarked in the Introduction, however, artifact systems are typically implemented through declarative languages such as GSM (Hull et al., 2011). It is therefore of interest to investigate the verification problem, not just on a Kripke semantics such as AC-MAS, but on actual programs. As discussed, while GSM is a mainstream declarative language for artifact-centric environments, alternative declarative approaches exist. In what follows for the sake of generality we ground our discussion on a very wide class of declarative languages and define the notion of *artifact-centric program*. Intuitively, an artifact-centric program (or AC program) is a declarative description of a whole multi-agent system, i.e., a set of services, that interact with the artifact system (see the discussion in the Introduction). Since artifact systems are also typically implemented declaratively (Heath et al., 2013), AC programs will be used to encode both the artifact system itself and the agents in the system. This also enables us to import into the formalism the previously discussed features of views and windows typical in GSM and other languages.

The rest of this section is organised as follows. We begin in Subsection 4.1 by defining AC programs and giving their semantics in terms of AC-MAS. We then show that any AC-MAS that results from an AC program is uniform. As long as the generated AC-MAS is bounded, by using the results of Section 3.3, we deduce that any AC program admits an AC-MAS as its finite model. In this context it is important to give constructive procedures for the generation of the finite abstraction; we provide such a procedure here. This enables us to state that, under the assumptions we identify, AC programs admit decidable verification by means of model checking their finite model. In Section 4.2 we ground and exemplify the constructions above on the Order-to-Cash Scenario introduced in Subsection 2.4.

4.1 Verifying Artifact-Centric Programs

We start by defining the abstract syntax of AC programs.

Definition 4.1 (AC Programs) *An* artifact-centric program *(or AC program) is a tuple* $ACP = \langle \mathcal{D}, U, \Sigma, \Psi \rangle$*, where:*

- $\mathcal{D}$ *is the program's database schema;*
- U *is the program's interpretation domain;*
- $\Sigma = \{\Sigma_0, \dots, \Sigma_n\}$ *is the set of* agent programs $\Sigma_i = \langle \mathcal{D}_i, l_{i0}, Act_i \rangle$*, where:*
  - $\mathcal{D}_i \subseteq \mathcal{D}$ *is* agent i's database schema*;*
  - $l_{i0} \in \mathcal{D}_i(U)$ *is* agent i's initial state (as a database instance)*;*
  - Act_i *is a set of local actions* $\alpha(\vec{x})$*, where* α *is the action name and* $\vec{x}$ *are the action parameter names; without loss of generality, we assume that no two action types use the same parameter names;*
  - *each local action* $\alpha(\vec{x})$ *is associated with a precondition* $\pi_{\alpha(\vec{x})}(\vec{y})$*, i.e., an FO-formula over* $\mathcal{D}_i$*, where* $\vec{y} \subseteq \vec{x}$ *are free variables.*
- $\Psi = \{\psi_{\vec{\alpha}(\vec{x})}(\vec{z}) \mid \vec{\alpha}(\vec{x}) = \langle \alpha_1(\vec{x}_1), \dots, \alpha_n(\vec{x}_n) \rangle \in Act_1 \times \cdots \times Act_n, \vec{x} = \langle \vec{x}_1, \dots, \vec{x}_n \rangle, \vec{z} \subseteq \vec{x}\}$ *represents the AC program's* transitions *expressed as a set of* postconditions*, i.e., FO-formulas over action parameters as free variables. The formulas in* Ψ *are defined over the alphabet* $\mathcal{D}_{Ag} \cup \mathcal{D}'_{Ag}$*, where* $\mathcal{D}_{Ag} = \{P_i^j/q_i \mid P_i/q_i \in \mathcal{D}, j \leq n\}$ *is the set containing one distinct relation symbol* P_i^j*, for each agent* $j \leq n$ *and relation symbol* $P_i \in \mathcal{D}$*.*

AC programs are defined modularly by giving all the agents' programs, including their action preconditions and postconditions. Notice that preconditions use relation symbols from the local database only, while the program's transitions Ψ refer to the local relations for all agents in an unconstrained way. More precisely, postconditions are *global*, i.e., they are associated with global actions, rather than local ones. Indeed, a formula $\psi_{\vec{\alpha}(\vec{x})}(\vec{z})$ describes the effects of the execution of the global action $\vec{\alpha}(\vec{z})$ (under a particular assignment to the parameters) where each agent i executes $\alpha_i(\vec{z}_i)$. As reported below, this accounts for the intuition that in choosing the next action, an agent can only rely on information locally stored, while its actions, as a result of mutual interactions, may change the local state of any agent, i.e., they affect the global state of the system. Obviously, this does not prevent the possibility of specifying actions that affect local states only. This is in line with the AC-MAS semantics and the literature on interpreted systems.

Given a tuple $\vec{x}$ of variables and a tuplue $\vec{u}$ of elements from U such that $|\vec{x}| = |\vec{u}|$, by $\sigma(\vec{x}) = \vec{u}$ we denote an assignment that binds the i-th component of $\vec{u}$ to the i-th component of $\vec{x}$. For a joint action $\vec{\alpha}(\vec{x})$ given as above, we let $con(\vec{\alpha}) = \bigcup_{i \leq n} con(\pi_i) \cup con(\psi)$, $var(\vec{\alpha}) = \bigcup_{i \leq n} var(\pi_i) \cup$

var(ψ), and *free*$(\vec{\alpha}) = \vec{x}$. An *execution* of $\vec{\alpha}(\vec{x})$ with *ground parameters* $\vec{u} \in U^{|\vec{x}|}$ is the *ground* action $\vec{\alpha}(\vec{u})$, where $\vec{v}$ (resp. $\vec{w}$) is obtained by replacing each y_i (resp. z_i) with the value occurring in $\vec{u}$ at the same position as y_i (resp. z_i) in $\vec{x}$. Such replacements make both each $\pi_i(\vec{v})$ and $\psi(\vec{w})$ ground, that is, first-order sentences. Finally, we define the set Con_{ACP} of all constants mentioned in the AC program ACP, i.e., $Con_{ACP} = \bigcup_{i=1}^{n} adom(l_{i0}) \cup \bigcup_{\vec{\alpha} \in Act} con(\vec{\alpha})$.

The semantics of an AC program is given in terms of the AC-MAS induced by the agents that the program implicitly defines. Formally, this is captured by the following definition.

Definition 4.2 (Induced Agents) *Given an AC program $ACP = \langle \mathcal{D}, U, \Sigma, \Psi \rangle$, an agent $A = \langle \mathcal{D}_i, Act_i, Pr_i \rangle$ is* induced *by ACP on the interpretation domain U iff for the agent program $\Sigma_i = \langle \mathcal{D}_i, l_{i0}, Act_i \rangle \in \Sigma$ we have that:*

- *for every $l_i \in \mathcal{D}_i(U)$ and ground action $\alpha(\vec{u})$ such that $\alpha(\vec{x}) \in Act_i$, it is the case that $\alpha(\vec{u}) \in Pr_i(l_i)$ iff $(l_i, \sigma) \models \pi_{\alpha(\vec{x})}(\vec{y})$, for $\sigma(\vec{x}) = \vec{u}$ (recall that $\vec{y} \subseteq \vec{x}$).*

Note that induced agents are agents as formalised in Definition 2.6. Agents defined as above are composed to give the AC-MAS associated with an AC program.

Definition 4.3 (Induced AC-MAS) *Given an AC program $ACP = \langle \mathcal{D}, U, \Sigma, \Phi \rangle$ and the set $Ag = \{A_0, \ldots, A_n\}$ of all agents induced by ACP, the AC-MAS* induced *by ACP is the tuple $\mathcal{P}_{ACP} = \langle Ag, s_0, \tau \rangle$, where:*

- $s_0 = \langle l_{00}, \ldots, l_{n0} \rangle$ *is the* initial global state*;*
- *τ is the global transition function defined by the following condition: $s' \in \tau(s, \vec{\alpha}(\vec{u}))$, where $s = \langle l_0, \ldots, l_n \rangle$, $s' = \langle l'_0, \ldots, l'_n \rangle$, $\vec{\alpha}(\vec{u}) = \langle \alpha_1(\vec{u}_0), \ldots, \alpha_n(\vec{u}_n) \rangle$, $\vec{u} = \langle \vec{u}_0, \ldots, \vec{u}_n \rangle$, iff for every $i \in \{0, \ldots, n\}$,*
  - $(l_i, \sigma_i) \models \pi_{\alpha_i(\vec{x}_i)}(\vec{y}_i)$ *for* $\sigma_i(\vec{x}_i) = \vec{u}_i$*;*
  - $adom(s') \subseteq adom(s) \cup \vec{u} \cup con(\psi_{\vec{\alpha}(\vec{x})})$*;*
  - *$(D_{Ag} \oplus D'_{Ag}, \sigma) \models \psi_{\vec{\alpha}(\vec{x})}(\vec{z})$, for an assignment σ such that $\sigma(\vec{x}) = \vec{u}$, and D_{Ag}, D'_{Ag} are the $\mathcal{D}_{Ag}$-instances such that, for every $P_i \in \mathcal{D}$ and $j \leq n$, $D_{Ag}(P_i^j) = l_j(P_i)$ and $D'_{Ag}(P_i^j) = l'_j(P_i)$.*

Given an AC program ACP, its induced AC-MAS represents the program's execution tree and encodes all the data in the system. Intuitively, this is obtained by iteratively executing at each state, starting from the initial one, all possible ground actions. Observe that all actions performed are enabled by the respective protocols and that transitions can introduce only a bounded number of new elements in the active domain, i.e., those bound to the action parameters. It follows from the above that AC programs are *parametric* with respect to the interpretation domain, i.e., by replacing the interpretation domain we obtain a different AC-MAS.

We assume that every program induces an AC-MAS whose transition relation is serial, i.e., states always have successors. This is a basic requirement that can be easily fulfilled, for instance, by assuming that each agent has a *skip* action with a trivially true precondition and that when all agents execute *skip*, the global state of the system remains unchanged. In the next Subsection we present an example of one such program.

A significant feature of AC programs is that they induce uniform AC-MAS.

Lemma 4.4 *Every AC-MAS $\mathcal{P}$ induced by an AC program ACP is uniform.*

Proof. Since by definition $adom(s_0) \subseteq Con_{ACP}$, by Prop. 3.7 it is sufficient to consider only the temporal transition relation $\rightarrow$. Consider $s, s', s'' \in \mathcal{S}$ and $s''' \in L_0 \times \cdots \times L_n$ such that $s \oplus s' \simeq s'' \oplus s'''$ for some witness ι. In particular, for every $A_j \in Ag$, $l''_j \simeq l_j$ and $l'''_j \simeq l'_j$, namely, $l''_j = \iota(l_j)$ and $l'''_j = \iota(l'_j)$. Also, assume that there exists $\vec{\alpha}(\vec{u}) = \langle \alpha_1(\vec{u}_1), \ldots, \alpha_n(\vec{u}_n) \rangle \in Act(U)$ such that $s' \in \tau(s, \vec{\alpha}(\vec{u}))$. First of all, $\alpha_j(\vec{u}_j) \in Pr_j(l_j)$ implies that $(l_j, \sigma_j) \models \pi_{\alpha_j(\vec{x}_j)}(\vec{y}_j)$ for $\sigma_j(\vec{x}_j) = \vec{u}_j$. Since $l''_j \simeq l_j$, by Prop. 3.3 we have that $(l''_j, \sigma'_j) \models \pi_{\alpha_j(\vec{x}_j)}(\vec{y}_j)$ where $\sigma'_j(\vec{x}_i) = \iota'(\vec{u}_i)$ for any ι' extending ι to $\vec{u}_i$. Thus, $\alpha_j(\iota'(\vec{u}_j)) \in Pr_j(l''_j)$ for every $j \in Ag$. Further, assume that $(D_{Ag} \oplus D'_{Ag}, \sigma) \models \psi_{\vec{\alpha}(\vec{x})}(\vec{z})$, where D_{Ag}, D'_{Ag} are the $\mathcal{D}_{Ag}$-instances obtained as above and $\sigma(\vec{x}) = \vec{u}$. Consider the $\mathcal{D}_{Ag}$-instances D''_{Ag}, D'''_{Ag} such that $D''_{Ag}(P^j_i) = l''_j(P_i) = \iota(l_j(P_i))$, and $D'''_{Ag}(P^j_i) = l'''_j(P_i) = \iota(l'_j(P_i))$. Since $s \oplus s' \simeq s'' \oplus s'''$, we obtain that $D_{Ag} \oplus D'_{Ag} \simeq D''_{Ag} \oplus D'''_{Ag}$ for the same witness ι. In particular, $(D''_{Ag} \oplus D'''_{Ag}, \sigma') \models \psi_{\vec{\alpha}(\vec{x})}(\vec{z})$, where $\sigma'(\vec{x}) = \iota'(\vec{u})$ for any ι' extending ι to $\vec{u}$. Finally, it can be easily checked that $adom(s''') \subseteq adom(s'') \cup \iota'(\vec{u}) \cup con(\psi_{\vec{\alpha}(\vec{x})})$. As a result, $s''' \in \tau(s'', \vec{\alpha}(\iota'(\vec{u})))$, i.e., $\mathcal{P}$ is uniform. $\square$

We can now define what it means for an AC program to satisfy a specification, by referring to its induced AC-MAS.

Definition 4.5 *Given an AC program ACP, a FO-CTLK formula φ, and an assignment σ, we say that ACP* satisfies *φ* under *σ, written $(ACP, \sigma) \models \varphi$, iff $(\mathcal{P}_{ACP}, s_0, \sigma) \models \varphi$.*

Thus, the model checking problem for an AC program against a specification ϕ is defined in terms of the model checking problem for the corresponding AC-MAS $\mathcal{P}_{ACP}$ against ϕ.

The following result allows us to reduce the verification of any AC program with an infinite interpretation domain U_1, that induces a b-bounded AC-MAS, to the verification of an AC program over a finite U_2. To show how this is constructed, we let $N_{ACP} = \sum_{i \in \{1,\ldots,n\}} \max_{\alpha(\vec{x}) \in \Omega_i} \{|\vec{x}|\}$ be the maximum number of different parameters that can occur in a joint action of ACP.

Lemma 4.6 *Consider an AC program $ACP_1 = \langle \mathcal{D}, U_1, \Sigma \rangle$ operating on an infinite interpretation domain U_1 and assume that its induced AC-MAS $\mathcal{P}_{ACP_1} = \langle Ag_1, s_{10}, \tau_1 \rangle$ is b-bounded. Consider a finite interpretation domain U_2 such that $Con_{ACP_1} \subseteq U_2$ and $|U_2| \geq 2b + |Con_{ACP_1}| + N_{ACP_1}$ and the AC program $ACP_2 = \langle \mathcal{D}, U_2, \Sigma \rangle$. Then, the AC-MAS $\mathcal{P}_{ACP_2} = \langle Ag_2, s_{20}, \tau_2 \rangle$ induced by ACP_2 is a finite abstraction of $\mathcal{P}_{ACP_1}$.*

Proof. Let Ag_1 and Ag_2 be the set of agents induced respectively by ACP_1 and ACP_2, according to Def. 4.2. Firstly, we prove that the set Ag_1 and Ag_2 of agents satisfy Def. 3.14, for $Ag = Ag_1$ and $Ag' = Ag_2$. To this end, observe that because ACP_1 and ACP_2 differ only in U, by Def. 4.2, $\mathcal{D} = \mathcal{D}'$, and $Act' = Act$. Thus, only requirement 3 of Def. 3.14 needs to be checked. For this, fix $i \in \{1, \ldots, n\}$ and assume that $\alpha(\vec{u}) \in Pr_i(l_i)$. By Def. 4.2, we have that $(l_i, \sigma) \models \pi_{\alpha_i(\vec{x}_i)}(\vec{y}_i)$ for $\sigma(\vec{x}_i) = \vec{u}_i$. By the assumption on $|U_2|$, since $con(\alpha) \subseteq Con_{ACP_1} \subseteq U_2$, $|\vec{u}| \leq N_{ACP_1}$, and $|adom(l_i)| \leq b$, we can define an injective function $\iota : adom(l_i) \cup \vec{u} \cup Con_{ACP_1} \mapsto U_2$ that is the identity on Con_{ACP_1}. Thus, for $l'_i = \iota(l_i)$, we can easily extract from ι a witness for $l_i \simeq l'_i$. Moreover, it can be seen that $\sigma(y) = \vec{v}$ and $\sigma'(y) = \vec{v}' = \iota(\vec{v})$ are equivalent for π. Then, by applying Prop. 3.3 to l_i and l'_i, we conclude that $(l'_i, \sigma') \models \pi_{\alpha_i(\vec{x}_i)}(\vec{y}_i)$. Hence, by Def. 4.2, $\alpha(\vec{u}') \in Pr'_i(l'_i)$ for $\vec{u}' = \iota(\vec{u})$. So, we have shown the right-to-left part of requirement 3. The left-to-right part can be shown similarly and more simply since U_1 is infinite.

Thus, we have proven that $Ag = Ag_1$ and $Ag' = Ag_2$ are obtained as in Def. 3.14. Hence, the assumption on Ag and Ag' in Def. 3.15 is fulfilled. We show next that also the remaining requirements of Def. 3.15 are satisfied. Obviously, since Σ is the same for ACP_1 and ACP_2, by Def. 4.3, $s_{10} = s_{20}$, so the initial states of $\mathcal{P}_{ACP_1}$ and $\mathcal{P}_{ACP_2}$ are the same. It remains to show that the requirements on τ_1 and τ_2 are satisfied. We prove the right-to-left part. To this end, take two states $s_1 = \langle l_{10}, \ldots, l_{1n} \rangle$, $s'_1 = \langle l'_{10}, \ldots, l'_{1n} \rangle$ in $\mathcal{S}_1$ and a joint action $\vec{\alpha}(\vec{u}) = \langle \alpha_0(\vec{u}_0), \ldots, \alpha_n(\vec{u}_n) \rangle \in Act(U_1)$ such that $s'_1 \in \tau_1(s_1, \vec{\alpha}(\vec{u}))$. Consider $s_1 \oplus s'_1$. By the assumptions on U_2, there exists an injective function $\iota : adom(s_1) \cup adom(s'_1) \cup \vec{u} \cup Con_{ACP_1} \mapsto U_2$ that is the identity on Con_{ACP_1} (recall that $|adom(s_1)|, |adom(s'_1)| \leq b$). Then, for $s_2 = \langle \iota(l_{10}), \ldots, \iota(l_{1n}) \rangle$, $s'_2 = \langle \iota(l'_{10}), \ldots, \iota(l'_{1n}) \rangle$ in $\mathcal{S}_2$, we can extract from ι a witness for $s_1 \oplus s'_1 \simeq s_2 \oplus s'_2$. Moreover, it can be seen that for every $\pi_{\alpha_i(\vec{x}_i)}$ and $\psi_{\vec{\alpha}(\vec{x})}$, the assignments $\sigma(\vec{x}) = \vec{u}$ and $\sigma'(\vec{x}) = \vec{u}' = \iota(\vec{u})$ are equivalent with respect to $s_1 \oplus s'_1$ and $s_2 \oplus s'_2$. Now, consider Def. 4.3 and recall that both $\mathcal{P}_{ACP_1}$ and $\mathcal{P}_{ACP_2}$ are AC-MAS induced by ACP_1 and ACP_2, respectively. By applying Prop. 3.3, we have that, for $i \in \{0, \ldots, n\}$, (i) $(\iota(l_{1i}), \sigma') \models \pi_{\alpha_i(\vec{x}_i)}(\vec{y}_i)$ iff $(l_{1i}, \sigma) \models \pi_{\alpha_i(\vec{x}_i)}(\vec{y}_i)$; (ii) $(D_{Ag_2} \oplus D'_{Ag_2}, \sigma') \models \psi_{\vec{\alpha}(\vec{x})}(\vec{z}_i)$ iff $(D_{Ag_1} \oplus D'_{Ag_1}, \sigma) \models \psi_{\vec{\alpha}(\vec{x})}(\vec{z}_i)$, where each D_{Ag_i} is obtained from s_i as detailed in Def. 4.3; (iii) $adom(s'_1) \subseteq adom(s_1) \cup \vec{u} \cup con(\psi_{\vec{\alpha}(\vec{x})})$ iff $adom(s'_2) \subseteq adom(s_2) \cup \iota(\vec{u}) \cup con(\psi_{\vec{\alpha}(\vec{x})})$ by the definition of ι. But then, it is the case that $s'_2 \in \tau_2(s_2, \vec{\alpha}(\iota(\vec{u})))$. So we have proved the right-to-left part of the second requirement of Def. 3.15. The other direction follows similarly. Therefore, $\mathcal{P}_{ACP_2}$ is an abstraction of $\mathcal{P}_{ACP_1}$. $\square$

Intuitively, Lemma 4.6 shows that the following diagram commutes, where $[U_1/U_2]$ stands for the replacement of U_1 by U_2 in the definition of ACP_1. Observe that since U_2 is finite, one can actually apply Def. 4.3 to obtain $\mathcal{P}_{ACP_2}$; in particular the transition function τ_2 can be computed. Instead, $\mathcal{P}_{ACP_1}$, and in particular τ_1, cannot be directly computed from ACP_1 by applying Def. 4.3, as U_1 is infinite.

$$\begin{tikzcd} ACP_1 \arrow[r, "\text{Def. 4.3}"] \arrow[d, "{[U_1/U_2]}"] & \mathcal{P}_{ACP_1} \arrow[d, "\text{Def. 3.15}"] \\ ACP_2 \arrow[r, "\text{Def. 4.3}"'] & \mathcal{P}_{ACP_2} \end{tikzcd}$$

The following result, a direct consequence of Lemma 3.17 and Lemma 4.6, is the key conclusion of this section.

Theorem 4.7 *Consider an FO-CTLK formula φ, an AC program ACP_1 operating on an infinite interpretation domain U_1 and assume its induced AC-MAS $\mathcal{P}_{ACP_1}$ is b-bounded. Consider a finite interpretation domain U_2 such that $C_{ACP_1} \subseteq U_2$ and $|U_2| \geq 2b + |C_{ACP_1}| + \max\{N_{ACP_1}, |var(\varphi)|\}$, and the AC program $ACP_2 = \langle \mathcal{D}, U_2, \Sigma \rangle$. Then we have that:*

$$ACP_1 \models \varphi \quad \textit{iff} \quad ACP_2 \models \varphi.$$

Proof. By Lemma 4.6 $\mathcal{P}_{ACP_2}$ is a finite abstraction of $\mathcal{P}_{ACP_1}$. Moreover, $|U_2| \geq 2b + |Con_{ACP_1}| + \max\{N_{ACP_1}, |var(\varphi)|\}$ implies $|U_2| \geq 2b + |Con_{ACP_1}| + |var(\varphi)|$. Hence, we can apply Lemma 3.17 and the result follows. $\square$

The results shows that if the generated AC-MAS model is bounded, then any AC program can be verified by model checking its finite abstraction, i.e., a bisimilar AC-MAS defined on a finite interpretation domain. Note that the procedure is constructive: given an AC program $ACP_1 = \langle \mathcal{D}, U_1, \Sigma \rangle$ on an infinite domain U_1 and an FO-CTLK formula φ, to check whether ACP_1 satisfies the specification φ, we first consider its finite abstraction $ACP_2 = \langle \mathcal{D}, U_2, \Sigma \rangle$ defined on a finite domain U_2 satisfying the cardinality requirement of Theorem 4.7. Since U_2 is finite, the induced AC-MAS $\mathcal{P}_{ACP_2}$ is also finite; hence we can apply standard model checking techniques to verify whether $\mathcal{P}_{ACP_2}$ satisfies φ. Finally, by definition of satisfaction for AC programs and Theorem 4.7, we can transfer the result obtained to decide the model checking problem for the original infinite AC program ACP_1 against the specification φ.

Also observe that in the finite abstraction considered above the abstract interpretation domain U_2 depends on the number of distinct variables that the specification φ contains. Thus, in principle, to check the same AS program against a different specification φ', one should construct a new abstraction $\mathcal{P}_{ACP_2'}$ using a different interpretation domain U_2', and then check φ' against it. However, it can be seen that if the number of distinct variables of φ' does not exceed that of φ, the abstraction $\mathcal{P}_{ACP_2}$, used to check φ, can be re-used for φ'. Formally, let FO-CTLK$_k$ be the set of all FO-CTLK formulas containing at most k distinct variables. We have the following corollary to Theorem 4.7.

Corollary 4.8 *If $|U_2| \geq 2b + |Con_{ACP_1}| + \max\{N_{ACP_1}, k\}$, then, for every FO-CTLK$_k$ formula φ, $ACP_1 \models \varphi$ iff $ACP_2 \models \varphi$.*

This result holds in particular for $k = N_{ACP}$; thus for FO-CTLK$_{N_{ACP}}$ formulas, we have an abstraction procedure that is specification-independent.

Theorem 4.7 requires the induced AC-MAS to be bounded, which may seem a difficult condition to check a priori. Note however that AC programs are declarative. It is therefore straightforward to give postconditions that enforce that no transition will generate states violating the boundedness requirement. The scenario in the next Subsection will exemplify this.

4.2 Verifying the Order-to-Cash Scenario

In Section 2.4 we introduced the order-to-cash scenario (Hull et al., 2011), a business process modelled as an artifact-centric system. Now we show how it can be formalised within the framework of AC programs. For the sake of simplicity we assumed only three agents in our scenario: one customer c, one manufacturer m and one supplier s. Further, the database schema $\mathcal{D}_i$ for each agent $i \in \{c, m, s\}$ was given as:

- Customer c: $\mathcal{D}_c = \{Products(pcode, budget), PO(id, pcode, offer, status)\}$;
- Manufacturer m: $\mathcal{D}_m = \{PO(id, pcode, offer, status), MO(id, pcode, price, status)\}$;
- Supplier s: $\mathcal{D}_s = \{Materials(mcode, cost), MO(id, pcode, price, status)\}$.

Also, we assumed that in the initial state the only non-empty relations are $Products$ and $Materials$. Hence, the artifact-centric program ACP_{otc} corresponding to the order-to-cash scenario can be given formally as follows:

Definition 4.9 **(**ACP_{otc}**)** *The* artifact-centric program ACP_{otc} *is a tuple* $\langle \mathcal{D}_{otc}, U_{otc}, \Sigma_{otc}, \Psi_{otc} \rangle$, *where:*

- *the program's database schema $\mathcal{D}_{otc}$ and interpretation domain U_{otc} are defined as in Sec. 2.4, i.e., $\mathcal{D}_{otc} = \mathcal{D}_c \cup \mathcal{D}_m \cup \mathcal{D}_s = \{PO/4, MO/4, Products/2, Materials/2\}$ and U_{otc} is the set of all alphanumeric strings.*

$\pi_{createPO(id,pcode)}$	$= \exists b.Products(b, pcode) \wedge$ $\neg\exists p, o, s.PO(id, p, o, s)$	requires id to be a fresh identifier for POs, and the newly created PO to refer to an existing product
$\pi_{doneMO(id)}$	$= \exists pc, p.MO(id, pc, p, \mathsf{preparation})$	requires id to refer to an existing MO currently in preparation
$\pi_{acceptMO(id)}$	$= \exists pc, p.MO(id, pc, p, \mathsf{submitted})$	which requires id to refer to an existing MO that has been submitted

Table 1: Preconditions for the actions $createPO(id, pcode)$, $doneMO(id)$, and $acceptMO(id)$

- $\Sigma = \{\Sigma_c, \Sigma_m, \Sigma_s\}$ *is the set of* agent specifications *for the customer c, the manufacturer* m *and the supplier s. Specifically, for each* $i \in \{c, m, s\}$, $\Sigma_i = \langle \mathcal{D}_i, l_{i0}, Act_i, \Pi_i \rangle$ *is such that:*
  - $\mathcal{D}_i \subseteq \mathcal{D}$ *is agent i's database schema as detailed above, i.e.,* $\mathcal{D}_c = \{Products/2, PO/4\}$, $\mathcal{D}_m = \{PO/4, MO/4\}$, *and* $\mathcal{D}_s = \{MO/4, Materials/2\}$.
  - l_{c0}, l_{m0}, *and* l_{s0} *are database instances in* $\mathcal{D}_c(U_{otc})$, $\mathcal{D}_m(U_{otc})$, *and* $\mathcal{D}_s(U_{otc})$ *respectively s.t.* $l_{c0}(Products)$ *and* $l_{s0}(Materials)$ *are non-empty, i.e., they contain some background information, while* $l_{c0}(PO)$, $l_{m0}(PO)$, $l_{m0}(MO)$ *and* $l_{s0}(MO)$ *are all empty.*
  - *The sets of actions are given as*
    * $Act_c = \{createPO(id, pcode), submitPO(id), pay(id), deletePO(id), skip\}$.
    * $Act_m = \{createMO(id, price), doneMO(id), shipPO(id), deleteMO(id), skip\}$;
    * $Act_s = \{acceptMO(id), rejectMO(id), shipMO(id), skip\}$.

    Each action $\alpha(\vec{x})$ *is associated with a precondition* $\pi_{\alpha(\vec{x})}$. *The preconditions for the actions* $createPO(id, pcode)$, $doneMO(id)$, $acceptMO(id)$ *are reported in Table 1. The remaining preconditions are omitted for brevity.*

- $\Psi = \{\psi_{\vec{\alpha}(\vec{x})} \mid \alpha(\vec{x})) \in Act_c \times Act_m \times Act_s\}$, *where*

  $$\mathcal{D}_{Ag} = \{Products^c, PO^c, PO^m, MO^m, Materials^s, MO^s\}.$$

  Table 2 illustrates only the postcondition of the joint action

  $$\vec{\alpha}(id, pc, m_1, m_2) = \langle createPO(id, pc), doneMO(m_1), acceptMO(m_2) \rangle.$$

  The others are omitted.

In the postcondition in Table 2 variables (from V) and constants (from U) are distinguished by fonts v and c, respectively. The first two lines impose that the interpretation of the relations $Products$ and $Materials$, occurring only in the local database of agents c (customer) and s (supplier), respectively, remain unchanged. The third line states that the relation PO of agents c and m (manufacturer) contains a new procurement order, with identifier id and product code pc, both taken from the parameters of action $createPO$. Observe that, although executed by the customer, this action affects also the local state of the manufacturer. The next 3 lines express that the local PO relation of c and m, in addition to the newly added item, contains also all, and only, the items present before the action execution. The next conjunct (3 lines) states that new identifiers must be unique within each local PO relation. Notice that while this cannot be guaranteed by agent c when executing $createPO$

$$
\begin{aligned}
&\big(\forall \vec{x}.Products^c(\vec{x}) \leftrightarrow Products^{c\prime}(\vec{x})\big)\wedge\\
&\big(\forall \vec{y}.Materials^s(\vec{y}) \leftrightarrow Materials^{s\prime}(\vec{y})\big)\wedge\\
&\big(\exists b.Products^c(pc,b) \wedge PO^{c\prime}(id,pc,b,\mathsf{prepared}) \wedge PO^{m\prime}(id,pc,b,\mathsf{prepared})\big)\wedge\\
&\Big(\forall i,pc,b,s.i \neq id \rightarrow \big(\\
&\qquad\qquad (PO^{c\prime}(i,pc,b,s) \leftrightarrow PO^c(i,pc,b,s))\wedge\\
&\qquad\qquad (PO^{m\prime}(i,pc,b,s) \leftrightarrow PO^c(i,pc,b,s))\big)\Big)\wedge\\
&\Big(\forall i,pc,b,s,pc',b',s'.\big(\\
&\qquad\qquad (PO^{c\prime}(i,pc,b,s) \wedge PO^{c\prime}(i,pc',b',s') \rightarrow (pc=pc' \wedge b=b' \wedge s=s'))\wedge\\
&\qquad\qquad (PO^{m\prime}(i,pc,b,s) \wedge PO^{m\prime}(i,pc',b',s') \rightarrow (pc=pc' \wedge b=b' \wedge s=s'))\big)\Big)\wedge\\
&\Big(m_1 = m_2 \rightarrow \big(\forall m_3,pc,p,s.\\
&\qquad\qquad (MO^m(m_3,pc,p,s) \leftrightarrow MO^{m\prime}(m_3,pc,p,s))\wedge\\
&\qquad\qquad (MO^s(m_3,pc,p,s) \leftrightarrow MO^{s\prime}(m_3,pc,p,s))\big)\Big)\wedge\\
&\Big(m_1 \neq m_2 \rightarrow \big(\forall pc,p,s.MO^m(m_1,pc,p,s) \rightarrow (\\
&\qquad\qquad\qquad\qquad \neg MO^{m\prime}(m_1,pc,p,s) \wedge MO^{m\prime}(m_1,pc,p,\mathsf{submitted})\wedge\\
&\qquad\qquad\qquad\qquad \neg MO^{s\prime}(m_1,pc,p,s) \wedge MO^{s\prime}(m_1,pc,p,\mathsf{submitted}))\big)\wedge\\
&\qquad\qquad \big(\forall pc,p,s.MO^s(m_2,pc,p,s) \rightarrow (\\
&\qquad\qquad\qquad\qquad \neg MO^{s\prime}(m_2,pc,p,s) \wedge MO^{s\prime}(m_2,pc,p,\mathsf{accepted})\wedge\\
&\qquad\qquad\qquad\qquad \neg MO^{m\prime}(m_2,pc,p,s) \wedge MO^{m\prime}(m_2,pc,p,\mathsf{accepted}))\big)\Big)\wedge\\
&\Big(\forall m_3,pc,p,s.m_1 \neq m_2 \wedge m_1 \neq m_3 \rightarrow \big(\\
&\qquad\qquad (MO^{m\prime}(m_3,pc,p,s) \leftrightarrow MO^m(m_3,pc,p,s))\wedge\\
&\qquad\qquad (MO^{s\prime}(m_3,pc,p,s) \leftrightarrow MO^m(m_3,pc,p,s))\big)\Big)
\end{aligned}
$$

Table 2: The postcondition $\psi_{\vec{\alpha}(id,pc,m_1,m_2)}$ for the joint action $\vec{\alpha}(id,pc,m_1,m_2) = \langle createPO(id,pc), doneMO(m_1), acceptMO(m_2)\rangle$

(as it cannot access relation PO of m), this value might actually be returned automatically by the system, and then used as input by the agent. The successive 3 lines state that if m_1 and m_2 coincide, i.e., two distinct operations are to be executed on the same material order m_1, then the action has no effect on any local MO relation. On the contrary, as the successive 6 lines state, if $m_1 \neq m_2$ then in the local MO relations of both agent s and m the material order with id m_1 changes its state to submitted and the one with id m_2 to accepted. Finally, the last 3 lines state that all material orders not involved in the executed (joint) action are propagated unchanged to their respective local relations.

Notice that although actions are typically conceived to manipulate artifacts of a specific class, their preconditions and postconditions may depend on artifact instances of different classes. For example, note that the action $createMO$ manipulates MO artifacts, but its precondition depends on PO artifacts. Also, we stress that action executability depends not only on the $status$ attribute of an artifact, but on the actual data content of the whole database, i.e., of all the other artifacts. Similarly, action executions affect not only $status$ attributes. Most importantly, by using first-order formulas such as $\phi_b = \forall x_1, \ldots, x_{b+1} \bigvee_{i \neq j}(x_i = x_j)$ in the postcondition ψ, we can guarantee that the AC program in question is bounded and is therefore amenable to the abstraction methodology of Section 4.

We now define the agents induced by the AC program ACP_{otc} given above according to Definition 4.2.

Definition 4.10 *Given the AC program $ACP_{otc} = \langle \mathcal{D}_{otc}, U_{otc}, \Sigma_{otc} \rangle$, the agents A_c, A_m and A_s induced by ACP_{otc} are defined as follows:*

- *$A_c = \langle \mathcal{D}_c, Act_c, Pr_c \rangle$, where (i) $\mathcal{D}_c$ is as above; (ii) $Act_c = \Omega_c = \{createPO, submitPO, pay, deletePO\}$; and (iii) $\alpha(\vec{u}) \in Pr_c(l_c)$ iff $(l_c, \sigma) \models \pi_{\alpha(\vec{x})}(\vec{y})$ for $\sigma(\vec{x}) = \vec{u}$.*

- *$A_m = \langle \mathcal{D}_m, Act_m, Pr_m \rangle$, where (i) $\mathcal{D}_m$ is as above; (ii) $Act_m = \Omega_m = \{createMO, doneMO, shipPO, deleteMO\}$; and (iii) $\alpha(\vec{u}) \in Pr_m(l_m)$ iff $(l_m, \sigma) \models \pi_{\alpha(\vec{x})}(\vec{y})$ for $\sigma(\vec{x}) = \vec{u}$.*

- *$A_s = \langle \mathcal{D}_s, Act_s, Pr_s \rangle$, where (i) $\mathcal{D}_s$ is as above; (ii) $Act_s = \Omega_s = \{acceptMO, rejectMO, shipMO\}$; and (iii) $\alpha(\vec{u}) \in Pr_s(l_s)$ iff $(l_s, \sigma) \models \pi_{\alpha(\vec{x})}(\vec{y})$ for $\sigma(\vec{x}) = \vec{u}$.*

Note that the agents A_c, A_m and A_s strictly correspond to the agents defined in Def. 2.12. In particular, by the definition of A_m above we can see that $createMO(id, price) \in Pr_m(l_m)$ if and only if the interpretation $l_m(PO)$ of the relation *PO* in the local state l_m contains a tuple $\langle id, pc, o, \mathsf{prepared} \rangle$ for some product pc and offer o; while $doneMO(mo\_id) \in Pr_m(l_m)$ iff $l_m(MO)$ contains a tuple with id mo_id and status preparation. As a result, the formal preconditions for $createMO$ and $doneMO$ satisfy the intended meaning of these actions.

We can now define the AC-MAS generated by the set of agents $Ag = \{A_c, A_m, A_s\}$ according to Definition 4.3.

Definition 4.11 *Given the AC program ACP_{otc} and the set $Ag = \{A_c, A_m, A_s\}$ of agents induced by ACP_{otc}, the AC-MAS induced by ACP_{otc} is the tuple $\mathcal{P}_{otc} = \langle Ag, s^0_{otc}, \tau_{otc} \rangle$, where:*

- *$s^0_{otc} = \langle l_{c0}, l_{m0}, l_{s0} \rangle$ is the initial global state, where the only non-empty relations are $Products$ and $Materials$;*

- *τ_{otc} is the global transition function defined according to Def. 4.3.*

The AC-MAS generated by the AC program ACP_{otc} corresponds closely to the AC-MAS appearing in Def. 2.13. As an example we give a snippet of the transition function τ_{otc} by considering the global action $\alpha(\vec{u}) = \langle createPO(id, pcode), doneMO(m_1), acceptMO(m_2)\rangle$ enabled by the respective protocols in a global state s. By the definition of actions $createPO(id, pcode)$, $doneMO(m_1)$, and $acceptMO(m_2)$ we have that $l_i(s) \in Pr_i$ for $i \in \{c, m, s\}$ implies that the $Products$ relation contains information about the product $pcode$. Also, the interpretation of the relation MO contains the tuples $\langle m_1, p, pr, \mathsf{preparation}\rangle$ and $\langle m_2, p', pr', \mathsf{submitted}\rangle$ for some products p and p'. By the definition of τ_{otc} it follows that for every $s' \in \mathcal{S}_{otc}$, $s \xrightarrow{\alpha(\vec{u})} s'$ implies that $(D_{Ag} \oplus D'_{Ag}, \sigma) \models \psi_{\alpha(\vec{u})}(id, pcode, m_1, m_2)$, where D_{Ag} and D'_{Ag} are obtained from s and s' by renaming the relation symbols, $\alpha(\vec{u}) = \langle createPO(id, pcode), doneMO(m_1), acceptMO(m_2)\rangle$, and σ is an interpretation of the formal parameters id, $pcode$, m_1 and m_2 in U_{otc}. In particular, the interpretation of the relation PO in D'_{Ag} extends both $D_{Ag}(PO^c)$ and $D_{Ag}(PO^m)$ with the tuple $\langle id, pc, b, \mathsf{prepared}\rangle$, where id is a fresh identifier. The tuples for the material orders m_1 and m_2 are updated in $D'_{Ag}(MO^m)$ (resp. $D'_{Ag}(MO^s)$) and become $\langle m_1, p, pr, \mathsf{submitted}\rangle$ (resp. $\langle m_2, p', pr', \mathsf{accepted}\rangle$). In light of the specification of $\psi_{\alpha(\vec{u})}$ for action $\alpha(\vec{u})$, no other element is updated in the transition. Finally, notice that these extensions are indeed the interpretations of PO and MO in s'. Thus, the semantics satisfies the intended meaning of the actions. It can also be checked that, in line with the discussion in Section 2.4, a full version of the function τ_{otc} given above can easily encode the artifacts' lifecycles as given in Figure 2.

We now proceed to exploit the methodology of Section 4 to verify the AC program ACP_{otp}. We use the formula φ_{match} from Section 2.4 as an example specification; analogous results can be obtained for the other formulas. Observe that according to Definition 4.3 the AC-MAS induced by ACP_{otp} has infinitely many states. We assume two interpretations for the relations $Products$ and $Materials$, which determine an initial state D_0. Consider the maximum number max of parameters and the constants C_Ω in the operations in Ω_c, Ω_m and Ω_s. In the case under analysis we have that $max = 2$. We earlier remarked that formulas such as ϕ_b in the postcondition of actions force the AC-MAS $\mathcal{P}_{otc}$ corresponding to ACP_{otc} to be bounded. Here we have that $\mathcal{P}_{otc}$ is b-bounded. According to Corollary 3.19, we can therefore consider any finite domain U' such that

$$
\begin{array}{rcl}
U' & \supseteq & D_0 \cup C_\Omega \cup con(\varphi_{match}) \\
 & \supseteq & D_0(Products) \cup D_0(Materials) \cup C_\Omega
\end{array}
$$

and such that

$$
\begin{array}{rcl}
|U'| & \geq & 2b + |D_0| + |C_\Omega| + |con(\varphi_{match})| + max \\
 & = & 2b + |D_0| + |C_\Omega| + 2
\end{array}
$$

For instance, we can consider any subset U' of U_{otc} satisfying the conditions above. Given that U' satisfies the hypothesis of Theorem 4.7, it follows that the AC program ACP_{otc} over U_{otc} satisfies φ_{match} if and only if ACP_{otc} over U' does. But the AC-MAS induced by the latter is a finite-state system, which can be constructively built by running the AC program ACP_{otc} on the elements in U'. Thus, $ACP_{otc} \models \varphi_{match}$ is a decidable instance of model checking that can be therefore solved by means of standard techniques.

A manual check on the finite model indeed reveals that $\varphi_{match}, \varphi_{budget}$ and φ_{cost} are satisfied in the finite model, whereas φ_{fulfil} is not. By Corollary 3.19 the AC-MAS $\mathcal{P}_{otc}$ induced by ACP_{otp} satisfies the same specifications. Hence, in view of Definition 4.5, we conclude that the artifact-centric program ACP_{otp} satisfies $\varphi_{match}, \varphi_{budget}$ and φ_{cost} but does not satisfy φ_{fulfil}. This is in line with our intuitions of the scenario.

5. Conclusions and Future Work

In this paper we put forward a methodology for verifying agent-based artifact-centric systems. We proposed AC-MAS, a novel semantics incorporating first-order features, that can be used to reason about multi-agent systems in an artifact-centric setting. We observed that the model checking problem for these structures against specifications given in a first-order temporal-epistemic logic is undecidable and proceeded to identify a suitable fragment for which decidability can be retained. Specifically, we showed that the class of bounded, uniform AC-MAS we identified admit finite abstractions that preserve the first-order specification language we introduced. Previous results in the literature, discussed in Subsection 1.2, limit the preservation to fragments of the quantified language and do not allow the interplay between first-order quantifiers and modalities.

We explored the complexity of the model checking problem in this context and showed this to be EXPSPACE-complete. While this is obviously a hard problem, we need to consider that these are first-order structures which normally lead to problems with high complexity. We note that the abstract interpretation domain is actually linear in the size of the bound considered.

Mindful of the practical needs for verification in artifact-centric systems, we then explored how finite abstractions can actually be built. To this end, rather than investigating one specific data-centric language, we defined a general class of declarative artifact-centric programs. We showed that these systems admit uniform AC-MAS as their semantics. Under the assumption of bounded systems we showed that model checking these multi-agent system programs is decidable and gave a constructive procedure operating on bisimilar, finite models. While the results are general, they can be instantiated for various artifact-centric languages. For instance, Belardinelli et al. (2012b) explore finite abstractions of GSM programs by using these results.

We exemplified the methodology put forward on a use case consisting of several agents purchasing and delivering products. While the system has infinitely many states we showed it admits a finite abstraction that can be used to verify a variety of specifications on the system.

A question left open in the present paper is whether the uniform condition we provided is tight. While we showed this to be a sufficient condition, we did not explore whether this is necessary for finite abstractions or whether more general properties can be given. In this context it is of interest that artifact-centric programs generate uniform structures. Also, it will be worthwhile to explore whether a notion related to uniformity can be applied to other domains in AI, for example to retain decidability of specific calculi. This would appear to be the case as preliminary studies in the Situation Calculus demonstrate (De Giacomo, Lespérance, & Patrizi, 2012).

On the application side, we are also interested in exploring ways to use the results of this paper to build a model checker for artifact-centric MAS. Previous efforts in this area (Gonzalez, Griesmayer, & Lomuscio, 2012) are limited to finite state systems. It would therefore be of great interest to construct finite abstractions on the fly to check practical e-commerce scenarios such as the one here discussed.

References

Abiteboul, S., Hull, R., & Vianu, V. (1995). *Foundations of Databases*. Addison-Wesley.

Alonso, G., Casati, F., Kuno, H. A., & Machiraju, V. (2004). *Web Services - Concepts, Architectures and Applications*. Data-Centric Systems and Applications. Springer.

Alves, A., Arkin, A., Askary, S., Barreto, C., Ben, Curbera, F., Ford, M., Goland, Y., Guízar, A., Kartha, N., Liu, C. K., Khalaf, R., König, D., Marin, M., Mehta, V., Thatte, S., van der Rijn, D., Yendluri, P., & Yiu, A. (2007). Web Services Business Process Execution Language Version 2.0. Tech. rep., OASIS Web Services Business Process Execution Language (WSBPEL) TC.

Bagheri Hariri, B., Calvanese, D., De Giacomo, G., Deutsch, A., & Montali, M. (2013). Verification of Relational Data-centric Dynamic Systems with External Services. In Hull, R., & Fan, W. (Eds.), *Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS'13)*, pp. 163–174. ACM.

Baresi, L., Bianculli, D., Ghezzi, C., Guinea, S., & Spoletini, P. (2007). Validation of Web Service Compositions. *IET Software*, *1*(6), 219–232.

Baukus, K., & van der Meyden, R. (2004). A knowledge based analysis of cache coherence. In Davies, J., Schulte, W., & Barnett, M. (Eds.), *Proceedings of the 6th International Conference on Formal Engineering Methods (ICFEM'04)*, Vol. 3308 of *Lecture Notes in Computer Science*, pp. 99–114. Springer.

Belardinelli, F., & Lomuscio, A. (2012). Interactions between Knowledge and Time in a First-Order Logic for Multi-Agent Systems: Completeness Results. *Journal of Artificial Intelligence Research*, *45*, 1–45.

Belardinelli, F., Lomuscio, A., & Patrizi, F. (2011a). A Computationally-Grounded Semantics for Artifact-Centric Systems and Abstraction Results. In Walsh, T. (Ed.), *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI'12)*, pp. 738–743. AAAI.

Belardinelli, F., Lomuscio, A., & Patrizi, F. (2011b). Verification of Deployed Artifact Systems via Data Abstraction. In Kappel, G., Maamar, Z., & Nezhad, H. R. M. (Eds.), *Proceedings of the 9th International Conference on Service-Oriented Computing (ICSOC'11)*, Vol. 7084 of *Lecture Notes in Computer Science*, pp. 142–156. Springer.

Belardinelli, F., Lomuscio, A., & Patrizi, F. (2012a). An Abstraction Technique for the Verification of Artifact-Centric Systems. In Brewka, G., Eiter, T., & McIlraith, S. A. (Eds.), *Proceedings of the 13th International Conference on Principles of Knowledge Representation and Reasoning (KR'12)*. AAAI.

Belardinelli, F., Lomuscio, A., & Patrizi, F. (2012b). Verification of gsm-based artifact-centric systems through finite abstraction. In Liu, C., Ludwig, H., Toumani, F., & Yu, Q. (Eds.), *Proceedings of the 10th International Conference on Service-Oriented Computing (ICSOC'12)*, Vol. 7636 of *Lecture Notes in Computer Science*, pp. 17–31. Springer.

Berardi, D., Calvanese, D., De Giacomo, G., Hull, R., & Mecella, M. (2005). Automatic Composition of Transition-based Semantic Web Services with Messaging. In Böhm, K., Jensen, C. S., Haas, L. M., Kersten, M. L., Larson, P.-Å., & Ooi, B. C. (Eds.), *Proceedings of the 31st International Conference on Very Large Data Bases (VLDB'05)*, pp. 613–624. ACM.

Berardi, D., Cheikh, F., Giacomo, G. D., & Patrizi, F. (2008). Automatic Service Composition via Simulation. *International Journal of Foundations of Computer Science*, *19*(2), 429–451.

Bertoli, P., Pistore, M., & Traverso, P. (2010). Automated Composition of Web Services via Planning in Asynchronous Domains. *Artificial Intelligence*, *174*(3-4), 316–361.

Bhattacharya, K., Gerede, C. E., Hull, R., Liu, R., & Su, J. (2007). Towards Formal Analysis of Artifact-Centric Business Process Models. In Alonso, G., Dadam, P., & Rosemann, M. (Eds.), *Proceedings of the 5th International Conference on Business Process Management (BPM'07)*, Vol. 4714 of *Lecture Notes in Computer Science*, pp. 288–304. Springer.

Blackburn, P., de Rijke, M., & Venema, Y. (2001). *Modal Logic*, Vol. 53 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press.

Calvanese, D., De Giacomo, G., Lenzerini, M., Mecella, M., & Patrizi, F. (2008). Automatic Service Composition and Synthesis: the Roman Model. *IEEE Data Engineering Bulletin*, *31*(3), 18–22.

Ciobaca, S., Delaune, S., & Kremer, S. (2012). Computing Knowledge in Security Protocols Under Convergent Equational Theories. *Journal of Automated Reasoning*, *48*(2), 219–262.

Clarke, E. M., Grumberg, O., & Peled, D. A. (1999). *Model Checking*. The MIT Press.

Cohn, D., & Hull, R. (2009). Business Artifacts: A Data-Centric Approach to Modeling Business Operations and Processes. *IEEE Data Engineering Bulletin*, *32*(3), 3–9.

Damaggio, E., Deutsch, A., & Vianu, V. (2012). Artifact Systems with Data Dependencies and Arithmetic. *ACM Transactions on Database Systems*, *37*(3), 22:1–22:36.

Damaggio, E., Hull, R., & Vaculín, R. (2011). On the Equivalence of Incremental and Fixpoint Semantics for Business Artifacts with Guard-Stage-Milestone Lifecycles. In Rinderle-Ma, S., Toumani, F., & Wolf, K. (Eds.), *Proceedings of the 9th International Conference on Business Process Management (BPM'11)*, Vol. 6896 of *Lecture Notes in Computer Science*, pp. 396–412. Springer.

De Giacomo, G., Lespérance, Y., & Patrizi, F. (2012). Bounded Situation Calculus Action Theories and Decidable Verification. In Brewka, G., Eiter, T., & McIlraith, S. A. (Eds.), *Proceedings of the 13th International Conference on Principles of Knowledge Representation and Reasoning (KR'12)*. AAAI.

Dechesne, F., & Wang, Y. (2010). To Know or not to Know: Epistemic Approaches to Security Protocol Verification. *Synthese*, *177*(Supplement-1), 51–76.

Deutsch, A., Hull, R., Patrizi, F., & Vianu, V. (2009). Automatic Verification of Data-centric Business Processes. In Fagin, R. (Ed.), *Proceedings of the 12th International Conference on Database Theory (ICDT'09)*, Vol. 361 of *ACM International Conference Proceeding Series*, pp. 252–267. ACM.

Deutsch, A., Sui, L., & Vianu, V. (2007). Specification and Verification of Data-Driven Web Applications. *Journal of Computer and System Sciences*, *73*(3), 442–474.

Fagin, R., Halpern, J. Y., Moses, Y., & Vardi, M. Y. (1995). *Reasoning About Knowledge*. The MIT Press.

Gammie, P., & van der Meyden, R. (2004). MCK: Model Checking the Logic of Knowledge. In Alur, R., & Peled, D. (Eds.), *Proceedings of 16th International Conference on Computer Aided Verification (CAV'04)*, Vol. 3114 of *Lecture Notes in Computer Science*, pp. 479–483. Springer.

Gerede, C. E., & Su, J. (2007). Specification and Verification of Artifact Behaviors in Business Process Models. In Krämer, B. J., Lin, K.-J., & Narasimhan, P. (Eds.), *Proceedings of the 5th International Conference on Service-Oriented Computing (ICSOC'07)*, Vol. 4749 of *Lecture Notes in Computer Science*, pp. 181–192. Springer.

Gonzalez, P., Griesmayer, A., & Lomuscio, A. (2012). Verifying GSM-Based Business Artifacts. In Goble, C. A., Chen, P. P., & Zhang, J. (Eds.), *Proceedings of the 19th IEEE International Conference on Web Services (ICWS'12)*, pp. 25–32. IEEE.

Grohe, M. (2001). Generalized Model-Checking Problems for First-Order Logic. In Ferreira, A., & Reichel, H. (Eds.), *Proceedings of the 18th Annual Symposium on Theoretical Aspects of Computer Science (STACS'01)*, Vol. 2010 of *Lecture Notes in Computer Science*, pp. 12–26. Springer.

Heath, F. T., Boaz, D., Gupta, M., Vaculín, R., Sun, Y., Hull, R., & Limonad, L. (2013). Barcelona: A Design and Runtime Environment for Declarative Artifact-Centric BPM. In Basu, S., Pautasso, C., Zhang, L., & Fu, X. (Eds.), *Proceedings of the 11th International Conference on Service-Oriented Computing (ICSOC'13)*, Vol. 8274 of *Lecture Notes in Computer Science*, pp. 705–709. Springer.

Hull, R. (2008). Artifact-Centric Business Process Models: Brief Survey of Research Results and Challenges. In Meersman, R., & Tari, Z. (Eds.), *Proceedings (part II) of Confederated International Conferences, CoopIS, DOA, GADA, IS, and ODBASE 2008 (On the Move to Meaningful Internet Systems: OTM'08)*, Vol. 5332 of *Lecture Notes in Computer Science*, pp. 1152–1163. Springer.

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. (2011). Business Artifacts with Guard-Stage-Milestone Lifecycles: Managing Artifact Interactions with Conditions and Events. In Eyers, D. M., Etzion, O., Gal, A., Zdonik, S. B., & Vincent, P. (Eds.), *Proceedings of the 5th ACM International Conference on Distributed Event-Based Systems (DEBS'11)*, pp. 51–62. ACM.

Hull, R., Narendra, N. C., & Nigam, A. (2009). Facilitating Workflow Interoperation Using Artifact-Centric Hubs. In Baresi, L., Chi, C.-H., & Suzuki, J. (Eds.), *Proceedings of the 7th International Conference on Service-Oriented Computing (ICSOC-ServiceWave '09)*, Vol. 5900 of *Lecture Notes in Computer Science*, pp. 1–18. Springer.

Kacprzak, M., Nabialek, W., Niewiadomski, A., Penczek, W., Pólrola, A., Szreter, M., Wozna, B., & Zbrzezny, A. (2008). VerICS 2007 - a Model Checker for Knowledge and Real-Time. *Fundamenta Informaticae*, *85*(1-4), 313–328.

Kupferman, O., Vardi, M. Y., & Wolper, P. (2000). An Automata-Theoretic Approach to Branching-Time Model Checking. *Journal of the ACM*, *47*(2), 312–360.

Lomuscio, A., Penczek, W., Solanki, M., & Szreter, M. (2011). Runtime Monitoring of Contract Regulated Web Services. *Fundamenta Informaticae*, *111*(3), 339–355.

Lomuscio, A., Qu, H., & Raimondi, F. (2009). MCMAS: A Model Checker for the Verification of Multi-Agent Systems. In Bouajjani, A., & Maler, O. (Eds.), *Proceedings of the 21st International Conference on Computer Aided Verification (CAV'09)*, Vol. 5643 of *Lecture Notes in Computer Science*, pp. 682–688. Springer.

Lomuscio, A., Qu, H., & Solanki, M. (2012). Towards Verifying Contract Regulated Service Composition. *Autonomous Agents and Multi-Agent Systems*, *24*(3), 345–373.

Lomuscio, A., & Raimondi, F. (2006). The Complexity of Model Checking Concurrent Programs Against CTLK Specifications. In Baldoni, M., & Endriss, U. (Eds.), *Proceedings of the 4th International Workshop on Declarative Agent Languages and Technologies (DALT'06), Selected, Revised and Invited Papers*, Vol. 4327 of *Lecture Notes in Computer Science*, pp. 29–42. Springer.

Marin, M., Hull, R., & Vaculín, R. (2013). Data Centric BPM and the Emerging Case Management Standard: A Short Survey. In La Rosa, M., & Soffer, P. (Eds.), *Proceedings of Business Process Management Workshops - BPM 2012 International Workshops. Revised Papers*, Vol. 132 of *Lecture Notes in Business Information Processing*, pp. 24–30. Springer.

Meyer, J.-J. C., & van der Hoek, W. (1995). *Epistemic Logic for AI and Computer Science*, Vol. 41 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press.

Nigam, A., & Caswell, N. S. (2003). Business Artifacts: An Approach to Operational Specification. *IBM Systems Journal*, *42*(3), 428–445.

Nooijen, E., Fahland, D., & Dongen, B. V. (2013). Automatic Discovery of Data-Centric and Artifact-Centric Processes. In La Rosa, M., & Soffer, P. (Eds.), *Proceedings of Business Process Management Workshops - BPM 2012 International Workshops. Revised Papers*, Vol. 132 of *Lecture Notes in Business Information Processing*, pp. 316–327. Springer.

Papadimitriou, C. H. (1994). *Computational complexity*. Addison-Wesley.

Parikh, R., & Ramanujam, R. (1985). Distributed Processes and the Logic of Knowledge. In Parikh, R. (Ed.), *Logic of Programs*, Vol. 193 of *Lecture Notes in Computer Science*, pp. 256–268. Springer.

Singh, M. P., & Huhns, M. N. (2005). *Service-Oriented Computing: Semantics, Processes, Agents*. John Wiley & Sons.

Wooldridge, M. (2000). Computationally Grounded Theories of Agency. In *Proceedings of the 4th International Conference on Multi-Agent Systems (ICMAS'00)*, pp. 13–22. IEEE.

Wooldridge, M. (2001). *Introduction to Multiagent Systems*. John Wiley & Sons.