ACSI – Artifact-Centric Service Interoperation

SEVENTH FRAMEWORK PROGRAMME

D2.1.2
Artifact verification and synthesis techniques
Iteration II

Project Acronym	ACSI	
Project Title	Artifact-Centric Service Interoperation	
Project Number	257593	
Workpackage	2	Formal-based techniques and tools
Lead Beneficiary	Imperial College of Science, Technology and Medicine	
Editor(s)	Francesco Belardinelli	Imperial
	Alessio Lomuscio	Imperial
	Nick Bezhanishvili	Imperial
Contributors(s)	Francesco Belardinelli	Imperial
	Alessio Lomuscio	Imperial
	Nick Bezhanishvili	Imperial
	Giuseppe De Giacomo	Uniroma
	Fabio Patrizi	Uniroma
	Riccardo De Masellis	Uniroma
	Paolo Felli	Uniroma
	Babak Bagheri Hariri	Bolzano
	Diego Calvanese	Bolzano
Reviewer	Marlon Dumas	UT
	Dirk Fahland	TU/e
Dissemination Level		Public
Contractual Delivery Date	30/05/2012	
Actual Delivery Date	30/05/2012	
Version	1.2	

The research leading to these results has received funding from the European Community's Seventh Framework Programme [FP7/2007-2013] under grant agreement no. 257593

The research leading to these results has received funding from the European Community's Seventh Framework Programme [FP7/2007-2013] under grant agreement no. 257593

Abstract

This document summarises the results obtained within Task 2.1 "Artifact Verification and Synthesis Techniques – Iteration II". Specifically, we present the requirements to be fulfilled in order to develop effective procedures for the verification and synthesis of artifact systems, we review the relevant state-of-the-art and present our original contributions in this line. These include the development of a grounded semantics for the model checking of artifact systems, the study of the model checking problem, abstraction techniques for the effective verification of artifact-systems, and an application to the verification of GSM-based artifact-systems. All original results included were or are to be presented in leading international conferences including ICSOC, IJCAI and KR.

Document History

Version	Date	Comments
0.1	08-05-2011	First draft
1.1	24-05-2011	Reviewers' changes implemented

Table of Contents

List of Tables

List of Figures

Acronyms

Acronym	Explanation
FO	First Order
WP	Work Package
MC	Model Checking
MAS	Multi-Agent System
ACS	Artifact-Centric System
QIS	Quantified Interpreted System

1 Introduction

This document describes the achievements of Task 2.1 (T2.1) "Artifact Verification and Synthesis Techniques" at the conclusion of Work Package 2 (WP2). The ultimate goal of the task is to develop a range of solid techniques for the verification and synthesis of artifact systems.

By verification of an artifact system we mean the following: given a formal model describing (a suitable abstraction of) the evolution of an artifact system, one is able to check whether some properties of interest are satisfied by this model, and hence by the system it represents. Properties of interest span from simple ones, such as reachability of a particular state (e.g., checking if the system can reach a state where no artifact is present), to more complex ones, such as checking that whenever an artifact is added, the system will eventually reach a particular status, or that the data content of the system at each step always satisfies some (relational) constraint, or even epistemic properties requiring that a participant has no strategy to access some sensitive information.

By synthesis we mean the process of (automatically) constructing coordinating mechanisms that make the artifacts of an artifact system interoperate. Moreover, this interoperation should be organized in such a way that properties similar to those mentioned above are satisfied. Similarly to the propositional case (which is much simpler), verification and synthesis techniques of artifact systems share the very same theoretical framework, and verification techniques provide useful guidelines for the development of synthesis procedures. Therefore, a good understanding of verification in the artifact context can be seen as a necessary *prerequisite* to tackle the harder issue of synthesis.

Two fundamental challenges were faced in order to accomplish T2.1. Both problems were successfully resolved. Firstly, formal models for artifact systems were developed. Secondly, the corresponding formalisms for the specification of the properties of interest have been adopted. As expected, we encountered a trade-off between richness of the model and expressivity of the specification formalism, and the complexity of the verification techniques. That is, the closer the model to the actual artifact system and the more expressive the specification formalism, the harder the algorithm to check the properties against the model.

Having in mind the objective of T2.1, our efforts have focused on identifying promising models and formalisms to capture interesting verification problems, as well as on investigating their computational properties, in terms of decidability and complexity. In doing so, we have taken into consideration the outcome of WP1, where, in particular, a general abstract model for artifact systems has been proposed. We introduced artifact models that are compliant with the GSM-based specification of artifact systems as proposed by WP1.

The distinguishing feature of the GSM model developed in WP1, as largely anticipated in preliminary analyses, is the presence of data and its infiniteness, which enables the systems to deal with infinite-states. This clearly constitutes a major obstacle to both verification and synthesis, as even simple problems easily become undecidable in this setting. This feature and the fact that the data organisation is an aspect of interest—e.g., one may be interested in whether each artifact instance, at some point, has a name and is associated to at least another artifact instance—are the main ingredients that make the verification and synthesis problems challenging. The results available in the literature do not provide actual techniques that can be successfully employed *off-the-shelf*, nor easily adapted to solve them.

Now we give a quick overview how the current document is organized. We start in Section 2 by identifying a core of requirements to be fulfilled in order to develop a framework and corresponding techniques for formal verification and synthesis in this context. This was achieved by extending the standard requirements of classical (propositional) verification of finite-state systems to the case of infinite-state systems with data, essentially relying on previous work (cf., e.g., [CGP99, DSV07]) and past experience [LQR09b].

After identifying these requirements, we proceed in Section 3 by reviewing related work. In particular, we discuss the work on verification and synthesis of finite- and infinite-state systems, and on abstraction techniques that was very useful in both narrowing the search space (thus providing a further solution to the *state-explosion* problem) and reducing a verification problem from infinite- to finite-state, thus guaranteeing decidability. Moreover, we briefly discuss the critical points in the ACSI project that could not be directly tackled by the existing techniques.

After this, we describe our main contributions. First, in Section 4 we adapt a framework for model checking multi-agent systems, namely that of *Quantified Interpreted Systems* [BL09], to the context of artifact systems, thus obtaining *Artifact-centric Multi-agent Systems* (AC-MAS). AC-MAS constitute a natural framework for reasoning about the evolution of an artifact system, as well as about epistemic properties involving the agents. This work shows how to use AC-MAS to capture verification problems on artifact (infinite-state) systems. It also provides a classification of abstraction methodology, while isolating cases of practical interest where the problem can be reduced to finite-state (hence decidable) model checking.

In Section 5 we validate our approach by using AC-MAS to formalise the GSM models in WP1. We then obtain decidability results for model checking GSM models.

The contribution described in Section 6, focuses on verification of temporal properties over data, without epistemic modalities. It develops a technique to detect whether an artifact system, specified by its data schema and exposed actions, deals with a bounded number of new elements, when a given process is executed over it. Whenever this is the case, the verification problem can be reduced to the finite case and is therefore addressed by means of (possibly adapted) standard model checking techniques. Importantly, this work does not focus on a particular temporal fragment, but considers the (very expressive) modal μ-calculus over *relational states.*

Section 7 considers the problem of synthesizing an agent protocol satisfying LTL specifications for multiple, partially-observable environments. A sound and complete procedure for this problem is presented, which is shown to be computationally optimal from a theoretical complexity standpoint. While Section 6 appeared in D2.1.1, Section 4, 5 and 7 are original.

Finally, we draw conclusions from the obtained results.

2 Requirements

In this section, we discuss the requirements that our framework needs to fulfill in order to serve as a formal basis in the development of actual tools for verification and synthesis of artifact(-based) systems (see also D2.2.2).

Verification of artifact systems refers to the process of checking whether a given system satisfies some design requirements. Such requirements can be broadly divided into two classes:

Intra-artifact requirements, capturing properties related to the internal structure of a particular artifact instance. They can refer to the current state of the artifact instance, e.g., requiring that every value of a particular attribute in an instance refers to another attribute in the same artifact, or can be more complex and take into account the whole instance lifecycle, requiring, e.g., that all along its evolution, the artifact instance goes through some particular state infinitely often.

Inter-artifact requirements. More general than those above, requirements from this class capture properties involving the interaction among artifact instances. An example is checking that every sent message is eventually consumed, or that artifact instances of a particular type are indeed generated by some instance of another type.

Conformance checking is the task of verifying an artifact system against a set of intra- and inter-artifact requirements, meant to capture the intended lifecycle of artifact instances. This is of particular interest as, when successful, guarantees that a system actually behaves as intended. Indeed, one of the main goals of this WP is to *build a formal framework providing the theoretical foundations and the algorithmic support for conformance checking.* This suggests, as a first requirement, that our framework must provide:

R1 *appropriate formalisms and techniques able to capture and verify temporal properties.*

However, this is not enough. As it turns out from WP1, the states of an artifact system can be modeled as (sets of) relational database instances, whose evolution essentially depends on updates performed by external participants, often human operators. As a consequence, no assumption can be made, in general, on the domains of the new values introduced at each step, and, in particular, such domains can be infinite, thus giving raise to infinitely many (relational) states. Of course, such features need to be accounted for in our framework, which thus requires that:

R2 *the formalism for requirement specification be able to express relationships among data*;

R3 *the verification techniques have the ability to deal with some form of infinite-state systems, possibly through approximations to finite models.*

As to [R2], a natural choice is to consider specification languages that include appropriate fragments of first-order logic. Concerning [R3], that is clearly affected by the (expressive power of) the language above, it is clear that classical techniques developed for finite-state systems cannot be applied, in general. Moreover, it turns out that all non-trivial, sufficiently general problem formalizations are undecidable, so that no complete verification procedure can be developed for the general case. Approaches to address this issue include: restricting the class of requirements to verify, restricting the class of systems, or combinations of both. Under such restrictions, it is often possible carrying out an *abstraction* process to produce a finite representation, the *abstract system*, of the original, concrete, infinite system, that can then be processed using classical techniques for the finite case, so as to obtain relevant information on the infinite behavior of the concrete system (cf., e.g., [BCG+05, DSV07, DHPV09, DDV11]). This is the approach explored in the ACSI project, although others, in particular incomplete techniques that do not necessarily resort to abstraction, are certainly worth investigating.

We stress that dealing with infinite states is the main challenge for the verification and synthesis tasks. Similarly to previous work (cf., e.g., Section 3), we were able to develop effective techniques only under appropriate restrictions, or by relaxing some requirements (e.g., completeness) on the quality of the techniques. Nonetheless, under such restrictions procedures able to solve interesting cases have been developed.

Observe that there is no reason, in general, for restricting verification to conformance checking only. Indeed, once a framework for verification of intra- and inter-artifact requirements is set, it can be used for the verification of any requirement, even if not (directly) related to artifact lifecycles. Therefore, we refer to *verification* as the generic task of checking whether an artifact-system fulfills a given set of (inter- and/or intra-artifact) requirements, which includes conformance checking as a special case.

In addition to verifying natural requirements as those described above, which essentially concern dynamic properties internal to the system, we also want to take a more external perspective, and verify requirements that involve the knowledge each participant has about the system, and in particular the data content. Such an interest is motivated by the observation that, in many cases, access to artifact data may be restricted, e.g., for security reasons, so a participant may be able to access only a partial fragment of the data content, and this might affect its ability to carry out (some of) the operations it was originally expected to offer. For instance, if a human participant, in order to serve a two-way call, needs to send an e-mail to someone whose address is recorded in an artifact inaccessible to the participant, then the required operation cannot be performed, and this may result in an undesired evolution of the artifact that initiated the two-way call (that, possibly, considers the call unsuccessful, after a timeout has occurred). But also more involved requirements are conceivable. Assume an artifact system is designed to prevent a certain class of participants from accessing some information. Verifying that the design actually enforces this property is certainly of interest, and this calls for checking the requirement that participants of the "bad" class cannot act jointly to retrieve (any fragment of) the reserved information.

In order to allow for verifying such *epistemic* requirements, our theoretical framework needs to be able to express knowledge-related properties. Therefore, it needs:

R4 *a formalism for requirement specification able to express epistemic properties involving data*;

R5 *verification technique(s) able to verify (an interesting class of) epistemic properties expressible in the formalism above.*

As it turns out, [R4] is an extension of [R2], as epistemic properties refer to (relationships among) data, and, similarly, also the corresponding verification techniques ([R5]) extend [R4]. Observe, however, that verification procedures tailored on special classes of requirements, often more effective than general ones, can be developed.

Importantly, as the objective of this WP is the development of actual, effective techniques for the verification of artifact systems, it seems natural to consider formalisms for requirement specification that are *computationally grounded* [Woo00] on existing models –such as the abstract model developed in WP1– that are well-suited to describe the (possibly simplified) evolution of artifact systems. This essentially means that the semantics of formulas in the adopted language must be provided in terms of such concrete (computational) models.

R6 *The language for requirement specification must be computationally grounded on existing computational models of artifact systems.*

In particular, observe that when considering epistemic properties, the notion of *agent* (capturing participants that interact with the artifact system) is an indispensable part of the model, as it represents the subject of knowledge, while it can certainly be removed if knowledge-related properties are not of interest.

Finally, we aim at actually implementing the techniques developed in this WP. In order for instances of the verification problem to be encoded and provided in input to the verification tool,

R7 *both computational models and requirement formalisms must be amenable to actual modelling languages of future design.*

In this respect, since the implementation is expected to be based on the existing tool MCMAS [LQR09b] (see D2.2.2), the candidate language for models is ISPL (Interpreted Systems Programming Language), possibly enriched to support first-order features.

As to synthesis in the context of artifact systems, broadly speaking, it refers to the definition of control procedures able to interact with the clients of the artifact system, and to control the system so as to guarantee desired requirements. Such requirements typically include properties about the data content of the system, its evolution, and even the epistemic state of participants.

For instance, one may need to build a *mediator* process that, by interacting, on the one side, with the artifact system, provides, on the other side, participant with a customized access to the system, e.g., by enabling only a limited set of actions on artifacts, or restricting the access to some artifact's attribute. In other words, such a process would act as a *filter*.

Typically, synthesis is harder than verification. In the literature a notable example of such a gap is the case of linear-time temporal logic, where the verification problem is PSPACE-complete [CGP99], while the synthesis problem for the same logic has a doubly exponential bound [PR89a] (both in the size of the requirement specification). These problems have a tight connection in that synthesizing a system for a given requirement is successful if and only if the obtained system can be successfully verified against the requirement. Intuitively, synthesis can be seen as preparing a system for a successful verification test, thus suggesting that, similarly to the case of synthesis for temporal specifications, in order to actually devise effective synthesis techniques, we need that

R8 *the framework and techniques developed for formal verification should be amenable for use/extension to synthesis.*

Observe that this includes the formalism for requirement specification and its respective semantics, defined according to the requirements identified above. As a desirable side-effect, verification techniques provide useful guidelines on the development of synthesis procedures.

3 State of the Art

In this section, we review the existing contributions in formal verification, synthesis, and abstraction literature, that are relevant to ACSI. Moreover, we discuss the main ACSI requirements that make such achievements not fully applicable in our context. As it often happens, even if the existing results do not perfectly match all project requirements, thus are not straightforwardly applicable, they certainly represent a solid background for solving problems that arise from specific project needs.

3.1 Verification

Approaches to formal verification can be broadly divided into two categories, depending on whether the class of the systems verified is finite- or infinite-state. In the following we review the previous contributions that are most relevant to ACSI.

3.1.1 Finite-state systems

By *finite-state* systems we mean (real) systems with a finite, (possibly huge) space of possible configurations. Typical examples of such systems include traffic lights, communication protocols, and hardware systems. One of the most popular techniques for verification of finite-state systems is *model checking* [CGP99], a technique based on building a suitable mathematical description of the system, referred to as *model*, of the requirement to verify, i.e., the *specification*, and then executing a verification algorithm that checks whether the model satisfies the specification.

Formally, the model is described as a *Kripke structure*, or *transition system*, i.e., essentially, a graph whose nodes represent different system states and whose edges capture possible state transitions. Each state is associated with a finite set of *propositional labels*, each representing an atomic property satisfied by the current state. For instance, if in the traffic-light example one represents the colors of the lights currently displayed by means of propositions r(ed), y(ellow) and g(reen), then the state where the red light is displayed would be labelled by r, while the state where yellow and green are displayed at once would be labelled by y and g.

As for specifications, they are typically formulas expressed in (fragments of) the *computation tree logic* CTL* [CES86, EH86], a language that, interpreted on Kripke structures, allows for capturing temporal possibility and necessity of properties to happen over time, as the system evolves. For instance, in the traffic-light example, one may express properties such as "it is always the case that the red light is eventually displayed", or "when the green light is displayed, the yellow light is displayed next".

Despite their relative simplicity, CTL* and its most widely used fragments, the *branching-time* temporal logic CTL [BAPM83, EC80] and the *linear-time* temporal logic LTL [Pnu81], proved successful in capturing many specifications of interest in the practice, as witnessed by their affirmation in industrial contexts for verification of both hardware and software (real) system components (see, e.g., [CW96] for a list of notable examples). More general specifications of Kripke structures can be expressed in propositional μ-calculus [Koz83], a modal logic with least and greatest fixpoint operators, powerful enough to capture the whole CTL* [Dam94].

As one may expect, depending on the expressive power of the specification language, one gets different complexity bounds of the verification task. In particular, verification of CTL formulas is polynomial in the size of the Kripke structure and the formula [CES86], LTL verification is

PSPACE-complete [SC85], and so is CTL* [SC85]. For both LTL and CTL*, the best known algorithms run in exponential time wrt the size of the requirement specification, while for propositional μ-calculus, the best known algorithm runs in exponential time in the *alternation depth* [1] of the specification [BCJ+97], and polynomial time in the size of the Kripke structure.

Typically, the model of a system (i.e., the Kripke structure) is not provided explicitly, i.e., listing all of its nodes, transitions and labels, but in a compact way, using a suitable language that, when interpreted, *generates* the Kripke structure. By *compact*, we mean that the size of the representation, also referred to as the *program*, is polynomial in the number of propositions labeling the states.

However, the underlying Kripke structure may contain a number of states that is exponential in that of propositions. The main problem associated with this phenomenon is known as the *state explosion* problem. Due to this, the explicit search over the whole set of states represented explicitly makes (explicit) model checking unsuitable for many real systems that, although finite, are characterized by a huge number of states.

The great success of model checking is due to the existence of optimization techniques that make the verification algorithms practically effective. Some of these are based on representing the underlying Kripke structure in a *symbolic* way, by using *Ordered Binary Decision Diagrams* (OBDD) [Bry86], i.e., an efficient representation of Boolean formulas used to characterize states and transitions of the Kripke structure [BCM+92]. Such optimizations made possible the implementation of verification tools, known as *symbolic* model checkers, able to manipulate models large enough to capture real systems, and consequently allowing for formal verification of such systems. Examples of symbolic model checkers include SMV [2], NuSMV [3], CadenceSMV [4].

Other optimization strategies are based on *partial order reduction*, a technique that allows for performing the search only on a *representative* subset of the state space, by exploiting the fact that the execution order of some transitions leads to states that are equivalent with respect to the requirement under verification. Typically, such techniques are best suited for *concurrent* systems verification. *Explicit* model checkers include SPIN [5] and UPPAAL [6].

An experimental comparison of the performance of various tools in the verification of LTL specifications was carried out in [RV10], showing that, on such class of specifications, symbolic tools significantly outperform explicit ones.

So far, we considered verification and model checkers for temporal specifications, only. However, the verification approach based on checking a model against a specification applies in general to other contexts. Of particular interest to ACSI is *verification of multi-agent systems* (MAS) [Woo09] or, more specifically, model checking for *temporal-epistemic* logics [vdMS99, KLP04]. This essentially refers to verifying whether some agents, acting and interacting according to their *protocols*, satisfy a desired specification. Such specifications, differently from the ones above, not only express properties concerning the system's evolution, but also allow for expressing the knowledge state, whereby *epistemic*, of agents with respect to both other agents and the environment they act in, as well as the ability of agent *coalitions* to achieve temporal-epistemic goals. Examples include: "an agent knows that another agent knows something", "all agents know that a particular property holds at a given state", "agents 1, 2, and 3, can cooperate so as

[1] A measure depending on the syntactic structure of the formula, which reflects the number of nested fixpoints that need to be computed in order to verify the formula.

[2] `www.cs.cmu.edu/~modelcheck/smv.html`

[3] `nusmv.fbk.eu/`

[4] `www.kenmcmil.com/smv.html`

[5] `spinroot.com`

[6] `www.uppaal.com`

to guarantee a particular property to be always satisfied", "agents 1 and 2 can prevent agent 3 from knowing what they know".

The relevance of MAS verification, and in particular model checking, comes essentially from two observations. Firstly, tools and techniques for MAS model checking have been successfully employed in the verification of several scenarios involving services as basic components [LQS08a, LQSS07, LQS08b], thus showing that multi-agent frameworks can be successfully adopted to capture service-oriented architectures. Secondly, agents can be used to model the interaction among the stakeholders and the artifact system, thus allowing for the verification of properties concerning not only the state of the system, but also the behaviour of each stakeholder, along with its epistemic state. For instance, one may check whether a stakeholder can access some information provided by the system, or if it can cooperate with other stakeholders so as to force a particular evolution of the system.

As it turns out, the language for such specifications needs, of course, appropriate constructs, including temporal operators such as those found in CTL*, *cooperation modalities* [AHK02] that allow for considering agent coalitions, and epistemic operators [FHMV95] to express properties about agents' and coalitions' state of knowledge. Many languages can be used, depending on the class of specifications one needs to capture. Of course, generality has a cost. For instance, the well-known *Alternating-time temporal logic* (ATL*) [AHK02], which provides temporal operators and cooperation modalities, but no epistemic operators, is a very rich language, strictly more expressive than CTL*, and, as expected, computationally more demanding. Precisely, ATL* model checking is a 2EXPTIME-complete problem [AHK02]. For this reason, the ATL* fragment ATL, where temporal operators are required to occur paired with a cooperation modality, is often preferred, as it provides a good compromise between expressivity and complexity. Indeed, ATL model checking is PTIME-complete [AHK02] (specifically, polynomial in the size of the structure to verify and the size of the specification) thus "only" exponential in the compact description of the problem instance. This suggests that the specification language needs to be carefully chosen, so as to obtain enough expressive power to capture (most of) the specifications of interest, while guaranteeing feasibility of the verification task.

In this project, we focus our attention on the MAS verification tool MCMAS [7] [LQR09b], as a starting point for our implementation. It is a symbolic model checker able to verify MAS against temporal-epistemic specifications with cooperation modalities. Properties are specified in the logic ATLK, a suitable extension of ATL with epistemic operators and cooperation modalities [LQR09b]. Observe that being CTL subsumed by ATL, and being ATLK an extension of ATL, it turns out that MCMAS can also be employed as a classical model checker for reactive systems. This suggests that while the approaches are very similar in spirit, MAS verification is, in fact, a (non-trivial) extension of reactive system verification (observe also that ATL* subsumes full CTL* [AHK02]). Details about the use of MCMAS as a verification tool in the ACSI project are provided in D2.2.2.

We remark that, in general, verification techniques for finite-state systems may represent a viable option for the verification of infinite-state systems. Indeed, one can provide a finite, high-level description of an infinite-state system, by taking into account only a finite number of (propositional) properties of interest. In other words, one can construct a coarse-grained system model and execute the verification procedure on it. Of course, this approach does not guarantee all specifications of interest to be verifiable, but may provide answers in some cases of interest. This approach is the basic idea behind *abstraction*, i.e., the process of simplifying a

[7] http://www-lai.doc.ic.ac.uk/mcmas/

given model by only considering the aspects that are relevant to the particular property to be verified. Existing approaches to abstraction in verification are addressed later on in this Section.

3.1.2 Infinite-state systems

The main problem associated with the verification of infinite-state systems is the general impossibility of performing an exhaustive search in the state space. Classical algorithms for model checking finite-state systems essentially fail because there is no guarantee that a fixpoint is eventually reached after a finite number of iterations. In fact, it is easy to find examples of undecidable verification tasks for infinite-state systems (e.g., reachability of a halting state in a Turing Machine).

Infinite-state systems often occur in the practice, and their verification is certainly of great practical interest. For instance, any C or Java program is potentially infinite-state, and so do artifact-based systems, as artifacts may contain values from infinite domains.

In the last two decades, the scientific community has paid great attention to this problem. System models have been proposed to capture relevant properties of infinite-systems, together with appropriate formalisms for requirement specification, classes of decidable (and undecidable) problems have been identified, and corresponding algorithms have been developed.

Technically, the definition of the model checking problem for the infinite-state case is exactly the same as for the finite-state case, that is, given a transition system and a property over it, check whether the property holds in the transition system. However, as said above, the infiniteness of the transition system makes the algorithms for classical model checking essentially useless in the general case.

A number of solutions have been proposed to deal with state infiniteness.[8] Many of them are based on identifying interesting classes of transition systems, definable by suitable formalisms, and some respective classes of decidable properties. Two fundamental results in this respect are [Büc62] and [GS84], where properties represented in Monadic Second-order Logic (MSO) are shown decidable respectively over linear orders and complete binary trees (representing the transition systems structure).

From these, a series of more general results are derived, over more complex structures obtained by suitable manipulations of basic transition systems, for which decidability results are known. Examples of such manipulations include *transductions* [Cou94], *tree-iterations* [Wal02] and *unfoldings*, that can be intuitively thought of as operations that build transition systems out of transition systems, by preserving some regularities that can be exploited by the verification algorithms.

For instance, in [Cau02], starting from the infinite complete binary tree, a hierarchy of infinite transition systems whose MSO-theories (i.e., sets of MSO properties defined over them) are decidable is obtained by means of *MSO-definable interpretations* [Cou94] and *unfoldings*, these representing operations similar to those described above. Similarly, other results state that the tree-iteration transformation preserves decidability of MSO theories [Sem84, Wal02]. On the same line, decidability of MSO theories over transition systems obtained from pushdown automata or prefix-rewriting systems is also proved [Cau03, MS85]. Concerning pushdown systems, it is also worth mentioning the existence of symbolic techniques used for reachability analysis [BEM97], and model checking algorithms for CTL properties [Wal00].

[8]Part of the following discussion follows [MP06], where many results in the area are reported and summarised.

Another approach is *regular model checking* [BJNT00, JN00, BHV04], a uniform paradigm for algorithmic verification of several classes of parameterized and infinite-state systems. In this account system states are captured by strings of arbitrary length over a finite alphabet, and the transition relation is given by a regular, length-preserving relation on strings, usually represented by a finite-state transducer. The fundamental problem of computing the set of reachable states from a given initial configuration, or *reachability analysis*, is tackled by using two complementary techniques: an automata-theoretic construction, and a fixpoint computation. Differently from the approaches discussed above, regular model checking is in general incomplete. Observe that, although sufficient conditions on the transition relation that guarantee termination are identified [BJNT00], incomplete techniques are useful in the practice even without them. Indeed, an incomplete algorithm can be executed over a finite temporal horizon, and provide useful information in case of termination (while not allowing for concluding anything, in general, when it does not terminate).

Other proposals and studies on model checking of infinite-state systems do exist –such as [BG04], [FL02] and [KMM^{+}01], just to mention some, which tackle the problem from both the theoretical and the practical perspectives, and which are certainly relevant to ACSI as, ultimately, artifact-based systems can be seen as a particular class of infinite-state systems. However, a distinctive feature of artifact-based systems is the presence of data in each state and their central importance with respect to both the evolution of the system, as the transition relation depends, in general, on the actual data content of each state, and the properties of interest, as the specifications to verify require, in general, to observe the data content and its (relational) organization. Therefore, while, on the one hand, the existing approaches and techniques offer a solid background for the development of actual verification procedures, on the other hand, additional theoretical tools are needed in order to achieve a complete understanding of how data explicitly represented in system states should be treated. While these points will be discussed more in details later on, here we review the most relevant work on verification of (infinite-state) systems with explicit representation of data. This relatively recent and little explored area deals with systems whose states represent data organized in some way (e.g., relational), and whose evolution depends on the data content of each state, its structure, and possible data received in input at each step.

One of the earliest work on this topic is [AVFY00], where *relational transducers* are proposed as a computational model of e-commerce applications. These are abstract machines whose states are instances of a relational database, and that can interact with external actors by taking in input and possibly returning in output some relations. The evolution of the system, as well as its output, depends on the current state of the system (i.e., the current state of the database) and the input it receives. The relation transition is specified in a declarative, relational language (a restriction of FO), which essentially states the relationship among the current state, the input, the next state, and the output. Several problems of interest are defined on such model, the most relevant to ACSI probably being verification of temporal properties involving data evolution. Importantly, the expressiveness of the language adopted has a significant impact on the decidability of the verification tasks, so several restrictions are imposed. In particular, it is required that the state is able only to accumulate all inputs received up to the current state. While this clearly limits the expressive power of a relational transducer, called, under the above restrictions, *Spocus transducer*, the model is still able to enforce non-trivial temporal properties on the system's (finite) runs.

The Spocus transducer of [AVFY00] is generalised into the *Abstract state machine (ASM)* relational transducer in [Spi03]. In this new model, the limitations on the state evolution are weakened, by essentially allowing more general FO formulas to occur in the specifications of the transition relation, while preserving decidability of verifying a temporal formula (over infinite runs) built using first-order formulas as atomic blocks.

ASM transducers are, in turn, extended in [DSVZ06, DSV07], where an enriched model, called ASM^+ transducer, is introduced. The relevance of ASM^+ transducers essentially resides in their ability to capture *data-driven* Web applications and to enable verification of FO temporal properties, like those considered in [Spi03], under analogous restrictions. Such restrictions constrain the way that elements in the current state can be accessed, by essentially introducing a form of guarded quantification on both the specification of the transition relation and the property to verify.

All the above transducers are, in general, infinite-state, as the databases (although finite) can take values from an infinite interpretation domain, thus giving rise to an infinite number of (finite) instances, whereby the difficulty of the verification task follows. Importantly, the (constructive) decidability proof developed in [DSV07] shows how, by means of an abstraction process made possible by the guarded quantification restriction, the verification problem for an (infinite-state) ASM^+ transducer can be reduced to classical model checking over a finite structure, built starting from the original transducer and the property to verify. Notably, this reduction represents the core of a verification procedure actually implemented in a real system [DMS$^+$05].

The relevance of these works to the ACSI project becomes even more apparent in recent work [DHPV09] where a variant of the ASM^+ framework presented above is adapted and extended to capture so-called *data-centric business processes*, which essentially correspond to particular classes of artifact systems. Furthermore, recently, in [DDV11] a variant of the framework adopted in [DHPV09] is proposed, where the verification task has been proved decidable for (suitable classes of) artifact systems in presence of data dependencies and arithmetic operations.

As it turns out, while there exist a large pool of results about infinite-state systems where data is not explicitly considered, much (theoretical and practical) work is still needed to verify systems that store and exchange data with the environment, as it is the case for artifact systems. Moreover, the work on the former focused, so far, on finding classes of systems and problems for which the verification task is decidable, and in particular allowing for a reduction to finite-state model checking, while no incomplete technique of practical relevance have been investigated yet, to the best of our knowledge. Finally, we remark the great relevance of exploring the connections between verification techniques that take data into account and those that do not, as it seems reasonable to expect the latter providing useful guidelines for the former.

3.2 Synthesis

Intuitively, *synthesis* is the problem of building a program that satisfies a given specification. Originally known as *Church's problem* [Chu62], in the last two decades the community of formal verification has paid great attention to this problem. Its importance becomes clear by observing that while verification allows for discovering flaws in a system *after* it has been built, synthesis allows for obtaining a system that is correct by construction, hence yielding a dramatic reduction of the effort needed to guarantee compliance of the system with the intended behaviour.

Open systems are central to synthesis. These are finite-state systems able to interact with the environment they act in, by reading and changing the (finite) values of a (finite) set of variables, over time. Such a model turns out to be quite expressive. For instance, it is able to model the evolution of a protocol that responds to incoming messages, or the behaviour of a lift that, upon receiving a request, changes its position accordingly, and becomes ready to serve new requests. An open system essentially captures the way a system reacts to the stimuli incoming from an external environment. Precisely, by *synthesis*, we refer to the construction of a *controller* module (typically finite-state) that, when interacting with the system (e.g., the protocol or the lift) in a *closed loop*, results in a behavior guaranteed to satisfy a desired temporal specification. Intuitively, we can think of the system as a car, the controller as the driver, and the desired specification as, e.g., "drive safely around a building".

Cornerstones in this area are [BL69] and [Rab72], where general solutions to Church's problem have been proposed, and, more directly related to this project, [PR89a] where a procedure for synthesising a controller (a.k.a. *reactive module*) starting from an LTL specification is provided. The obtained algorithm is based on checking emptiness of Rabin automata on infinite trees, and has a doubly exponential bound in the length of the specification. While representing a notable achievement from a theoretical point of view, the approach, due to its prohibitive complexity, turns out to be ineffective in practice.

Importantly, [PR89a] characterises synthesis as a form of verification on *game structures*, which are essentially transition systems along whose runs two types of states alternate: those representing the (open) system's actions, and those representing the environment's replies. The synthesis task then amounts to guaranteeing that the system has a strategy to fulfil the specification, no matter how the environment replies to its actions (along every run). The tight relation between synthesis and verification becomes even more apparent in the case of MAS verification where, e.g., verification of ATL formulas reduces, in fact, to a particular class of synthesis, in that the existence of a strategy to satisfy a specification is checked for (although the strategy itself is not computed).

To overcome the obstacle represented by the high problem complexity, and make synthesis for LTL specifications a task practically feasible, some efforts have been made towards identifying classes of more tractable specifications of practical interest [PPS06b, AMPS98, AT04]. In particular, [PPS06b], whose results can be seen as a generalisation of those in [AMPS98, AT04], presents an actual algorithm for the synthesis of so-called *generalised reactivity (1)* formulas (GR(1)), for which the synthesis task is singly exponential, and provides examples showing the practical interest of such class of specifications. This algorithm has been implemented (together with others) in the system TLV [9] (temporal logic verifier) and, recently, in its new, Java-based extension JTLV [PSZ10].

It is worth noticing that the above results served as basis for a series of works on the composition of behavioural services and agents [GMP09, GP09, GFPS10], where it is shown how the problem of building a desired behavior by suitably coordinating, through a synthesised controller, a set of given behaviours, can be reduced to an instance of the LTL synthesis problem. Based on the interpretation of the stakeholders as agents interacting with the artifact system, we believe that these results can provide a solid bases in tackling synthesis problems that involve the interaction between stakeholders and artifact system.

Finally, we mention [BCG$^+$05], which is the only result we are aware of about (a form of) synthesis in presence of data. This work presents a technique for constructing a *mediator* that, placed in the middle between a set of services (able to exchange messages) and a set of users, suitably coordinates the services so as to provide a new, desired service, not available otherwise. Similarly to those discussed above, this work, although developed in a different setting, could provide useful insights on the development of techniques for the synthesis of control procedures that guarantee stakeholders to interact with the system, while satisfying desired requirements

[9]`www.cs.nyu.edu/acsys/tlv/`

(e.g., access restrictions to particular artifacts).

3.3 Abstraction

Generally speaking, *abstraction* in verification is a technique for simplifying finite- or infinite-state systems by removing details that are not relevant to the property under verification. Typically, abstraction techniques amount to *clustering* states of the original system that satisfy some common properties into so-called *abstract states*, then deriving abstract transitions among such abstract states, and finally verifying the obtained (abstract) system. Different criteria defining how states are clustered and how the transition relation is derived, give raise to different abstraction approaches. However, all of them have in common the fact that since the abstract state space is typically smaller than the concrete space, the approach generally results in a significant reduction on the execution time of the verification algorithm, or in the possibility of verifying a property otherwise unverifiable, in the case of infinite-state systems. Of course, these advantages do not come, in general, for free, but require some limitations on the properties that can be verified on the abstract system.

Although abstraction as a mean for program verification dates back to the late 70's, with the foundational work by P. Cousot and R. Cousot [CC77], who proposed a general framework for the construction of sound approximations (abstractions) of mathematical structures, here we review more recent results, in particular those related to model checking, as directly relevant to the ACSI project.

One of the most popular abstraction approaches in model checking is described in [CGJ$^+$03] (that can be seen as an extension of the earlier *existential abstraction* framework presented in [CGL94]), where a technique, known as *counter-example guided abstraction refinement* (CEGAR), is presented. It is proved complete over the class of CTL* formulas that admit a single run as counterexample (namely, ACTL*), if the original system is finite-state.

The proposed algorithm intuitively works as follows:

1. an initial abstraction is derived from a compact specification (program) of the concrete model and the property under verification. In general, this step produces an abstract model that *over-approximates* the original model, that is, it contains *spurious* transitions not capturing any concrete transition;
2. the current abstract model is verified against the property of interest;
3. if the property is satisfied in the abstract model, it is guaranteed to be satisfied in the concrete model, too, and the process terminates successfully;
4. otherwise, the counterexample returned by the verification step is checked against the concrete model, to see if it captures any concrete behavior;
5. if it does, the process terminates, with the property of interest guaranteed not being satisfied by the concrete model;
6. if it does not, the current abstract model is refined by removing the spurious transitions, and a new iteration starts from step 2.

Even though the approach does not guarantee completeness for infinite-state systems, it still provides a viable, though incomplete, option in these cases. For instance, one can fix a bound on the number of iterations on the algorithm, and try to check the property of interest over the abstract models within the fixed bound. If the process terminates before reaching the bound, then the answer it provides is correct, while nothing can be said otherwise.

As discussed above (see Section 2), artifact system verification has many similarities with classical verification (of finite-state systems), in that the former can be seen as transition systems whose states have a particular (data) structure. Also, from our discussion above, it turned out that temporal-epistemic properties with first-order features, that can be seen as generalising classical temporal properties in model checking, are of interest for artifact systems. Moreover, the artifact-centric paradigm includes a formal, compact, representation of artifact systems, i.e., GSM or FSM, as discussed in WP1 report for M1.

These observations candidate the verification approach of [CGJ+03] as a possible starting point for the development of effective verification procedures for artifact systems. In particular, if an abstraction procedure for artifact systems were available that would guarantee, similarly to the propositional case, the preservation of interesting properties from the abstract to the concrete model, one could adapt the general structure of the CEGAR algorithm, so as to obtain an effective, though possibly incomplete, verification procedure for artifact systems.

However, two main obstacles make such an extension challenging: firstly, the presence of data, that makes the system infinite-state, and the relevance of its organization to the properties of interest, and, secondly, the interest in epistemic properties, that adds a further dimension to take into account in the verification procedure. As for the latter issue, efforts in that direction have already been undertaken. In particular, [CDLR09] adapts the existential abstraction approach proposed in [CGL94] to the context of *interpreted systems*, a well-known framework constituting the theoretical bases of MAS's and their corresponding (propositional) epistemic logics [FHMV95]. In Section 3.4, we further extend this work to the framework of artifact-systems, thus developing abstraction-based verification procedures in the artifact context. Such an extension finds its theoretical bases in recent work [BL09], where the standard (propositional) framework of interpreted systems, used to model multi-agent systems and to reason about agents knowledge, is extended to account for generic first-order properties. This results in a more general model that is particularly well-suited also for artifact systems (see Section 3.4 for details).

The abstraction approach of [CGJ+03] is not the only one proposed in the model checking context. Another similar approach, known as *predicate* (or *boolean*) abstraction, based on partitioning the state space of the concrete system according to relevant properties satisfied by each state is proposed in [GS97] and [Saï00]. Despite the approach being slightly more general, it achieves analogous results, in particular the preservation of ACTL* satisfaction from the abstract to the concrete level, the possibility of iteratively refining the abstract model by exploiting the obtained counterexamples, and the incompleteness result for infinite systems. Therefore, from the ACSI point of view, the challenges to be faced in order to suitably extend these approaches happen to be exactly the same.

The abstraction approaches described so far are tailored on propositional model checking. While their extension to the ACSI framework is certainly worth exploring, this is not the only available option to build effective verification procedures for artifact systems. Of course, (complementary) *ad-hoc* procedures tailored on artifact systems can also be developed. With respect to this, we mention again the series of works [DSVZ06, DSV07, DHPV09, DDV11], where abstraction techniques well-suited for systems that deal with explicit data are developed, that we believe can be adapted in the context of ACSI, and possibly extended to take the epistemic properties into account.

In Section 3.4, we describe a framework for the verification of temporal properties, involving data, over artifact systems. In such a work, an ad-hoc abstraction technique is developed to deal with new values that actor interacting with the system can possibly introduce as the system evolves.

3.4 Discussion

In this section, we have reviewed the literature on verification and synthesis of finite- and infinite-state systems that is most relevant to the ACSI project, and in particular to Task 2.1. This includes classical approaches for finite-state systems, such as model checking and synthesis for temporal properties, for infinite-state systems, and *ad-hoc* approaches to deal with data.

Since artifact systems can be seen as transition systems, we expect model checking approaches, both in the finite- and the infinite-state case, to provide useful insights about verification of artifact systems, even including the possibility of extending existing algorithms to deal with artifact systems.

However, we have identified two major obstacles that need to be overcome before MC-based techniques can become effective over artifact systems. Precisely, such obstacles arise from the presence of infinite data (and the fact that properties referring to the actual data content of the system are of interest), and the fact that such properties can be, in general, epistemic. Since verification of systems dealing with data is a little explored area, and even less verification of epistemic properties in presence of data, a major effort on the theoretical side is certainly required in order to understand how the problem can be effectively addressed.

Previous works of particular interest, which can serve as starting points in our investigation, include contributions on data-aware systems [DHPV09, DSV07] and abstraction techniques for MAS [CDLR09, BL09]. The former provide some bases for reducing MC over infinite systems that deal with data to MC over finite systems, while the latter both extend a popular abstraction technique to the framework of interpreted systems and introduce an extension of interpreted systems able to capture data in the state of the system. Ideas from these works can be successfully combined so as to build a new framework and actual procedures for the verification of temporal and temporal-epistemic properties over artifact systems.

Finally, we observe that existing approaches are typically aimed at identifying decidable classes of systems and/or properties that guarantee decidability. Considering that such decidable classes yield an (exponential) complexity bound that makes the problem, in general, unfeasible, it turns out that also incomplete techniques, possibly used in conjunction with complete ones, represent a further viable option to address the problem, certainly worth exploring.

4 An Abstraction Technique for the Verification of Artifact-Centric Systems

In this section we explore the verification problem for artifact-centric multi-agent systems, i.e., systems of agents interacting through artifact systems, by means of a knowledge-based perspective. We operate in a first-order setting with knowledge and branching time (FO-CTLK). We first remark that the general problem is undecidable and then proceed to give abstraction results that enable us, in a large class of cases of practical interest, to be able to analyse the model checking of finite approximations rather than the plain infinite model with unbounded database schemas.

The rest of the section is as follows. In Section 4.1 and following we identify the problem, give a general semantics for artifact-centric multi-agent systems (AC-MAS), as well as comment on the undecidability of the model checking problem in the general case. In Section 4.5 we explore abstraction results that enable us to translate the problem to the model checking of finite models. Relevant proofs are provided in the Appendix 9. Most of the material in this section has not appeared in D2.1.1, and has been published in [BLP11a, BLP11b, BLP12].

4.1 Databases and First-order Logic

We fix the terminology on databases that will be used in the rest of the section [AHV95a].

Definition 4.1 (Database schema)*:* A *(relational) database schema* is a set $\mathcal{D} = \{P_1/q_1, \ldots, P_n/q_n\}$ of *relation symbols* P_i, each associated with its arity $q_i \in \mathbb{N}$.

Interpretations of database schemas are defined over countable interpretation domains.

Definition 4.2 (Database interpretation)*:* Given a database schema $\mathcal{D}$, a *$\mathcal{D}$-interpretation (or $\mathcal{D}$-instance) over an interpretation domain 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 $\mathcal{D}$-interpretations over an interpretation domain U is denoted by $\mathcal{D}(U)$. The *active domain* of an interpretation 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 $adom(D)$.

Next, we briefly recall the syntax of first-order formulas with the equality symbol. Let Var be a countable set of *individual variables* and C be a finite set of *individual constants*. A *term* is any element $t \in Var \cup C$.

Definition 4.3 (FO-formulas over $\mathcal{D}$)*:* Given a database schema $\mathcal{D}$, the formulas φ in the first-order language $\mathcal{L}_{\mathcal{D}}$ are defined as follows:

$$\varphi \;\; ::= \;\; t = t' \mid P_i(t_1, \ldots, t_k) \mid \neg\varphi \mid \varphi \rightarrow \varphi \mid \forall x \varphi$$

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

$\mathcal{L}_{\mathcal{D}}$ is a standard first-order language over the relational vocabulary $\mathcal{D}$, with no function symbols other than constants, and with finitely many constant symbols from C. The fact that we consider a finite set of constants does not constitute a limitation. Indeed, as long as one deals with a finite set of formulas, as it is the case in our setting, C can always be extended so as to include all the (finitely many) constants occurring in such formulas. We use the standard abbreviations $\exists$, $\top$, $\bot$, $\wedge$, $\vee$, and $\neq$. *Free* and *bound* variables are defined as standard. For a formula φ we denote the set of its variables as $vars(\varphi)$, the set of its free variables as $free(\varphi)$, and the set of constants in φ as $const(\varphi)$. 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.

Assume an interpretation domain U such that $C \subseteq U$. 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 as the identity on constants, so that $\sigma(t) = t$, if $t \in C$. We can now define the semantics of $\mathcal{L}_{\mathcal{D}}$.

Definition 4.4 (Semantics of FO-formulas)*:* Given a $\mathcal{D}$-interpretation 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:

$$
\begin{array}{lll}
(D,\sigma) \models P_i(t_1,\ldots,t_{q_i}) & \text{iff} & \langle \sigma(t_1),\ldots,\sigma(t_{q_i})\rangle \in D(P_i) \\
(D,\sigma) \models t = t' & \text{iff} & \sigma(t) = \sigma(t') \\
(D,\sigma) \models \neg\varphi & \text{iff} & (D,\sigma) \not\models \varphi \\
(D,\sigma) \models \varphi \rightarrow \psi & \text{iff} & (D,\sigma) \not\models \varphi \text{ or } (D,\sigma) \models \psi \\
(D,\sigma) \models \forall x \varphi & \text{iff} & \text{for all } u \in adom(D),\ (D,\sigma\binom{x}{u}) \models \varphi
\end{array}
$$

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

Observe that we are adopting an *active-domain* semantics, that is, quantified variables range over the active domain of D. Also notice that constants are interpreted as themselves, i.e., two constants are equal if and only if they are syntactically the same. In the rest of the section, we assume that every interpretation domain includes C.

Finally, we introduce an operator on $\mathcal{D}$-instances that will be useful in the rest of the section. To this end, 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'.

Definition 4.5 ($\oplus$ Operator)*:* Given two $\mathcal{D}$-interpretations D and D', $D \oplus D'$ is the $(\mathcal{D} \cup \mathcal{D}')$-interpretation such that (i) $D \oplus D'(P_i) = D(P_i)$; and (ii) $D \oplus D'(P_i') = D'(P_i)$.

4.2 Artifact-Centric Multi-Agent Systems

We extend the definition of AC-MAS in [BLP12] so as to incorporate parametric actions. We start by introducing the notion of *agent*.

Definition 4.6 (Agent)*:* Let U be an interpretation domain. An *agent* over U is a tuple $A = \langle \mathcal{D}, U, L, Act, Pr \rangle$, where:

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

Agents can be thought of as modules that can be composed together to obtain *artifact-centric multi-agent systems* (AC-MAS) as defined below.

Definition 4.7 (Artifact-centric multi-agent system)*:* Given a set $Ag = \{A_0, \ldots, A_n\}$ of agents $A_i = \langle \mathcal{D}_i, U, L_i, Act_i, Pr_i \rangle$, an *artifact-centric multi-agent system* is a tuple $\mathcal{P} = \langle \mathcal{S}, U, s_0, \tau \rangle$, where:

- $\mathcal{S} \subseteq L_0 \times \cdots \times L_n$ is the set of *reachable global states*;
- U is the interpretation domain of $\mathcal{P}$;
- $s_0 \in \mathcal{S}$ is the *initial global state*;
- $\tau : \mathcal{S} \times Act(U) \mapsto 2^{\mathcal{S}}$ 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 iff $\alpha_i(\vec{u}_i) \in Pr_i(l_i)$ for $i \leq n$.

Usually, agent A_0 is referred to as the *environment* E. 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.

In the rest of the section we assume the function τ to be computable. In particular, for any state $s \in \mathcal{S}$ and ground action $\vec{\alpha}(\vec{u}) \in Act(U)$, we assume that $|\tau(s, \vec{\alpha}(\vec{u}))|$ is finite. This will allow us to abstract from the way τ is actually specified.

We define the *transition relation* $\rightarrow$ on $\mathcal{S} \times \mathcal{S}$ s.t. $s \rightarrow s'$ 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 $n \in \mathbb{N}$, $r(n) \doteq s^n$. 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, and that $\mathcal{S}$ is the set of states reachable from s_0. Two global states $s = \langle l_0, \ldots, l_n \rangle$ and $s' = \langle l'_0, \ldots, l'_n \rangle$ are said to be *epistemically indistinguishable* for agent A_i, written $s \sim_i s'$, if $l_i = l'_i$. This is consistent with the standard definition of knowledge as identity of local states [FHMV95]. Notice that when U is infinite, so is $\mathcal{S}$, and the AC-MAS $\mathcal{P}$ is an infinite-state system. Finally, we define the *global* database schema $\mathcal{D}$ of an AC-MAS as $\mathcal{D} = \mathcal{D}_0 \cup \cdots \cup \mathcal{D}_n$. 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 when it is clear from the context and we write $adom(s)$ for $adom(D_s)$. Notice that for any $s \in \mathcal{S}$ there exists a unique D_s, while the converse is not true in general.

4.3 Model Checking

We focus on the problem of verifying an artifact-centric multi-agent system against a temporal-epistemic specification of interest. Since the states of an artifact are characterised by their data content, the atomic components of the specifications need to capture relational properties pertaining to the states. This, together with the fact that the domain of data may be infinite, makes the problem substantially more challenging than standard model checking [CGP00].

We now define the first-order temporal epistemic specification language to be interpreted on AC-MAS.

Definition 4.8 (FO-CTLK)*:* The first-order 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 *vars*, *free*, and *const*. We use the abbreviations $EX\varphi$, $AF\varphi$, $AG\varphi$, $EF\varphi$, and $EG\varphi$ as standard. The epistemic formulas $K_i\varphi$ and $C\varphi$ are read as "agent A_i knows that φ" and "it is common knowledge that φ" respectively. Observe that free variables can occur within the scope of modal operators, thus allowing for the unconstrained alternation of quantifiers and modal operators. In what follows we consider also the *sentence atomic* fragment saFO-CTLK of FO-CTLK defined in the following BNF form:

$$\varphi \ ::= \ \phi \mid AX\varphi \mid A\varphi U\varphi \mid E\varphi U\varphi \mid K_i\varphi \mid C\varphi$$

where ϕ is a sentence in $\mathcal{L}_{\mathcal{D}}$ and $0 < i \leq n$. In saFO-CTLK no free variable occurs in the scope of a modal operator.

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

Definition 4.9 (Semantics of FO-CTLK formulas)*:* 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	$(\mathcal{P}, s, \sigma) \not\models \varphi$
$(\mathcal{P}, s, \sigma) \models \varphi \rightarrow \varphi'$	iff	$(\mathcal{P}, s, \sigma) \not\models \varphi$ or $(\mathcal{P}, s, \sigma) \models \varphi'$
$(\mathcal{P}, s, \sigma) \models \forall x\varphi$	iff	for all $u \in ad(D_s)$, $(\mathcal{P}, s, \sigma\binom{x}{u}) \models \varphi$
$(\mathcal{P}, s, \sigma) \models AX\varphi$	iff	for all r, if $r(0) = s$ then $(\mathcal{P}, r(1), \sigma) \models \varphi$
$(\mathcal{P}, s, \sigma) \models A\varphi\mathcal{U}\varphi'$	iff	for all r, if $r(0) = s$ then there is $k \geq 0$ s.t. $(\mathcal{P}, r(k), \sigma) \models \varphi'$, and for all j, $0 \leq j < k$ implies $(\mathcal{P}, r(j), \sigma) \models \varphi$
$(\mathcal{P}, s, \sigma) \models E\varphi\mathcal{U}\varphi'$	iff	for some r, $r(0) = s$ and there is $k \geq 0$ s.t. $(\mathcal{P}, r(k), \sigma) \models \varphi'$, and for all j, $0 \leq j < k$ implies $(\mathcal{P}, r(j), \sigma) \models \varphi$
$(\mathcal{P}, s, \sigma) \models K_i\varphi$	iff	for all s', $s \sim_i s'$ implies $(\mathcal{P}, s', \sigma) \models \varphi$
$(\mathcal{P}, s, \sigma) \models C\varphi$	iff	for all s', $s \sim^* s'$ implies $(\mathcal{P}, s', \sigma) \models \varphi$

where $\sim^*$ is the transitive closure of $\bigcup_{A_i \in Ag} \sim_i$.

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 $(\mathcal{P}, s_0) \models \varphi$.

In the rest of the section we explore model checking of AC-MAS against first-order temporal epistemic specifications. Formally, the problem can be stated as follows:

> Given an AC-MAS $\mathcal{P}$ and an FO-CTLK formula φ, find an assignment σ such that $(\mathcal{P}, s_0, \sigma) \models \varphi$.

Even in the context of a purely temporal language, the problem is decidable whenever the domain U is finite, and undecidable in general [BLP11b]. Since we are here operating on a strictly stronger language than the one of [BLP11b], the following result immediately follows.

Theorem 1. *The model checking problem for AC-MAS w.r.t. FO-CTLK is undecidable.*

This result is obviously negative. However, in this section we identify a large class of AC-MAS, covering interesting cases, for which model checking is decidable.

4.4 A Computationally-Grounded Semantics for Artifact-Centric Systems

We begin by reporting results from [BLP11a] in which we initially developed a grounded semantics for artifact centric systems as well as initial abstraction results. The semantics developed abstracts from database views (see Section 4.5) and is inspired by the semantics of quantified interpreted systems [BL09].

4.4.1 Simulation

The standard notion of simulation for reactive systems states that a system simulates another if every behaviour of the latter is a behaviour of the former [CGL94]. Since ACTL operators quantify over all behaviours (runs), any ACTL property that holds in the simulating system holds also in the simulated system. To extend this preservation property to FO-$\forall$ACTLK, i.e., the fragment of FO-CTLK containing universal modalities and the universal quantifier only, we require that any epistemic possibility in the simulated system is matched by an epistemic possibility in the simulating system. Similar conditions apply to individuals.

Definition 4.10 (Sublanguage)*:* Let $\mathcal{L}$ and $\mathcal{L}'$ be first-order temporal epistemic languages as in Def. 4.8, with alphabets $\mathcal{A}$ and $\mathcal{A}'$ respectively. $\mathcal{L}'$ is a sub-language of $\mathcal{L}$, or $\mathcal{L}' \subseteq \mathcal{L}$, if $\mathcal{A}' \subseteq \mathcal{A}$.

We now define the structures of interest and a notion of simulation.

Definition 4.11 (QIS)*:* A *quantified interpreted system* is tuple $\mathcal{P} = \langle \mathcal{S}, U, s_0, \tau, I \rangle$ s.t. $\mathcal{S}$, U, s_0 and τ are defined as for AC-MAS. Further, I is a first order interpretation for $\mathcal{L}$, that is, for every n-ary predicate symbol $P^n \in \mathcal{L}$, $I(P^n, s)$ is an n-ary relation on U.

Notice that, differently from the previous section, $\mathcal{L}$ can contain an infinite number of predicate symbols.

Definition 4.12 (Simulation)*:* Let $\mathcal{P} = \langle \mathcal{S}, U, s_0, \tau, I \rangle$ be a QIS on the set Ag of agents and the language $\mathcal{L}$, and let $\mathcal{P}' = \langle \mathcal{S}', U', s'_0, \tau', I' \rangle$ be a QIS on the same set Ag of agents and a sub-language $\mathcal{L}' \subseteq \mathcal{L}$. A simulation between $\mathcal{P}$ and $\mathcal{P}'$ is a pair of relations $\simeq \subseteq \mathcal{S} \times \mathcal{S}'$ and $\approx \subseteq U \times U'$ such that:

(a) $s_0 \simeq s'_0$;

(b) if $a \in U$ then there exists $a' \in U'$ such that $a \approx a'$;

and if $s \simeq s'$ then:

(c) if $s \to u$ then $s' \to' u'$ for some u' such that $u \simeq u'$;

(d) if $s \sim_i u$ then $s' \sim'_i u'$ for some u' such that $u \simeq u'$;

(e) for all $P^n \in \mathcal{L}'$, if $\vec{a} \approx \vec{a}'$ then $\vec{a} \in I(P^n, s)$ iff $\vec{a}' \in I'(P^n, s')$;

(f) for all $c \in \mathcal{L}'$, $I(c) \approx I'(c)$;

where $\to$ and $\to'$ are the transition relations in $\mathcal{P}$ and $\mathcal{P}'$ respectively. If there is a simulation pair between $\mathcal{P}$ and $\mathcal{P}'$, we say that $\mathcal{P}'$ simulates $\mathcal{P}$, or $\mathcal{P} \preceq \mathcal{P}'$.

According to (a) the initial state in $\mathcal{P}$ has to be matched by the initial state in $\mathcal{P}'$. Similarly for (b) and individuals. According to (c) and (d) every temporal and epistemic transition in $\mathcal{P}$

has to be matched by a transition in $\mathcal{P}'$. According to (e) and (f) related states must agree on the sub-language $\mathcal{L}'$.

Any FO-$\forall$ACTLK property is preserved from the simulating QIS $\mathcal{P}'$ to the QIS $\mathcal{P}$ being simulated:

Lemma 1. *Assume that $\mathcal{P}'$ simulates $\mathcal{P}$. For any FO-$\forall$ACTLK formula $\phi \in \mathcal{L}'$, if $\mathcal{P}' \models \phi$ then $\mathcal{P} \models \phi$.*

Differently from the propositional level, we can define a strengthening of the notion of simulation, so that also existential formulas are also preserved.

Definition 4.13 (Simulation$^+$)*:* A simulation$^+$ between $\mathcal{P}$ and $\mathcal{P}'$ is a pair of relations $\simeq \subseteq \mathcal{S} \times \mathcal{S}'$ and $\approx \subseteq U \times U'$ such that:

(a) $\simeq$ and $\approx$ are a simulation pair between $\mathcal{P}$ and $\mathcal{P}'$;

(b) if $a' \in U'$ then there exists $a \in U$ such that $a \approx a'$.

If there is a simulation$^+$ pair between $\mathcal{P}$ and $\mathcal{P}'$, we say that $\mathcal{P}'$ simulates$^+$ $\mathcal{P}$, or $\mathcal{P} \preceq^+ \mathcal{P}'$.

We can now prove the following strengthening of Lemma 1.

Lemma 2. *Assume that $\mathcal{P}'$ simulates$^+$ $\mathcal{P}$. For any FO-ACTLK formula $\phi \in \mathcal{L}'$, if $\mathcal{P}' \models \phi$ then $\mathcal{P} \models \phi$.*

In the rest of the paper we focus on simulation$^+$ and the FO-ACTLK fragment.

4.4.2 Existential abstraction of QIS

In systems with large state spaces, it is infeasible to verify design requirements by considering all reachable states, even if represented symbolically [BCM+92]. In existential abstraction [CGL94], one reduces a large, possibly infinite reactive system - referred to as the *concrete* system - into a possibly smaller reactive system - referred to as the *abstract* system - by partitioning the system states into equivalence classes. Each equivalence class, called an *abstract state*, forms a state in the abstract system.

Here, we extend existential abstraction to quantified interpreted systems by abstracting each agent i and the quantification domain U separately. Formally, the abstract QIS is defined as a quotient construction as follows. Assume a quantified interpreted system $\mathcal{P}$ over the set Ag of agents and the language $\mathcal{L}$. For each $i \in Ag$ assume the equivalence relations $\equiv_i \subseteq L_i \times L_i$ and $\equiv_i \subseteq ACT_i \times ACT_i$. Further, assume an equivalence relation $\equiv \subseteq U \times U$. For $l \in L_i$, write $[l]$ for the equivalence class of l w.r.t. $\equiv_i$, and $[\alpha]$ for the equivalence class of $\alpha \in ACT_i$ w.r.t. $\equiv_i$. Similarly, $[a]$ is the equivalence class of $a \in U$ w.r.t. $\equiv$. Write $[s]$ for $\langle [s_1], \ldots, [s_n] \rangle$ and write $[\vec{\alpha}]$ for $\langle [\alpha_1], \ldots, [\alpha_n] \rangle$. Finally, let $\mathcal{L}' \subseteq \mathcal{L}$ be a sub-language of $\mathcal{L}$ that does not distinguish between equivalent local states and individuals, i.e.,

(*) for all $P^n \in \mathcal{L}'$, if $s \equiv s'$ and $\vec{a} \equiv \vec{a}'$ then $\vec{a} \in I(P^n, s)$ iff $\vec{a}' \in I(P^n, s')$.

Definition 4.14 (Quotient QIS)*:* The quotient of $\mathcal{P}$ is the QIS $\mathcal{P}'$ on the set Ag of agents and the sub-language $\mathcal{L}' \subseteq \mathcal{L}$ such that:

1. $U' = \{[a] \mid a \in U\}$;
2. $L'_i = \{[l] \mid l \in L_i\}$;

3. $ACT'_i = \{[\alpha] \mid \alpha \in ACT_i\}$;
4. $P'_i = \{\langle [l], [\alpha] \rangle \mid \langle l, \alpha \rangle \in P_i\}$;
5. $\tau' = \{\langle [s], [\vec{\alpha}], [s'] \rangle \mid \langle s, \vec{\alpha}, s' \rangle \in \tau\}$;
6. $s'_0 = [s_0]$;
7. for all $P^n \in \mathcal{L}'$, $[\vec{a}] \in I'(P^n, [s])$ iff $\vec{a} \in I(P^n, s)$;
8. for all $c \in \mathcal{L}'$, $I'(c) = [I(c)]$.

Note that the interpretation I is well defined by condition (*) on the sub-language $\mathcal{L}'$. Observe that Def. 4.14 does not specify how the equivalence relations are chosen. This issue is addressed in Section 4.4.3 below. The important property of quotient systems, however, is that specifications are preserved from abstract systems to concrete ones. This is because the abstract system simulates$^+$ the original system.

Lemma 3. *If $\mathcal{P}'$ is a quotient of $\mathcal{P}$, then $\mathcal{P}'$ simulates$^+$ $\mathcal{P}$.*

Since the abstract system simulates$^+$ the concrete system, design requirements expressed in FO-ACTLK are preserved.

Theorem 2 (Preservation)**.** *Let $\mathcal{P}'$ be a quotient of the QIS $\mathcal{P}$. For any FO-ACTLK formula ϕ in $\mathcal{L}' \subseteq \mathcal{L}$, if $\mathcal{P}' \models \phi$ then $\mathcal{P} \models \phi$.*

The proof follows from Lemmas 2 and 3.

When applying the theorem, the challenge is to choose suitable equivalence relations $\equiv_i$ on local states and actions. We provide a partial answer in the following section.

4.4.3 Constructive abstraction

In this section we introduce a methodology for defining the equivalence relations in the previous section. In what follows we fix a sub-language $\mathcal{L}'$ of the first-order temporal epistemic language $\mathcal{L}_m$, and a QIS $\mathcal{P}$. We first define equivalences on the set U^n of n-tuples of individuals.

Definition 4.15 (Equivalence on tuples)*:* Let $s \in \mathcal{S}$ and let $\vec{a}$, $\vec{b}$ be n-tuples in U^n. We say that $\vec{a}$ and $\vec{b}$ are equivalent in s, or $\vec{a} \sim_s \vec{b}$, if for all $P^n \in \mathcal{L}'$, $\vec{a} \in I(P^n, s)$ iff $\vec{b} \in I(P^n, s)$.

We can easily check that the relation $\sim_s$ is indeed an equivalence relation for every $s \in \mathcal{S}$.

Definition 4.16 (Equivalence on individuals)*:* Let $s \in \mathcal{S}$ and let a, b be individuals in U. We say that a and b are equivalent in s, or $a \equiv_s b$, if for all $\vec{a}$, $\vec{a}' \in U^n$, if $\vec{a}'$ is obtained from $\vec{a}$ by uniformly substituting a with b, then $\vec{a} \sim_s \vec{a}'$.

From the fact that $\sim_s$ is an equivalence relation we can derive that also $\equiv_s$ is an equivalence relation for $s \in \mathcal{S}$.

We now define the equivalence relation on states.

Definition 4.17 (Equivalence on states)*:* Two states s, $s' \in \mathcal{S}$ are equivalent , or $s \equiv s'$, if for all $P^n \in \mathcal{L}'$, for all $\vec{a} \in U^n$, $\vec{a} \in I(P^n, s)$ iff $\vec{a} \in I(P^n, s')$.

We can now introduce the abstract model obtained from the QIS $\mathcal{P}$.

Definition 4.18 (Abstract model)*:* Given the QIS $\mathcal{P} = \langle \mathcal{S}, U, s_0, \tau, I \rangle$ we define an abstract model $\mathcal{M}' = \langle \mathcal{S}', U', [s_0], \tau', I' \rangle$ such that:

- $\mathcal{S}' = \{[s] \mid s \in \mathcal{S}\}$;
- for each $[s] \in \mathcal{S}'$, $U'([s]) = \{[a]_s \mid a \in U\}$;
- for any $[s], [s'] \in \mathcal{S}$, $[s] \to [s']$ if $s \to s'$;
- for $P^n \in \mathcal{L}'$, $[\vec{a}]_s \in I'(P^n, [s])$ iff $\vec{a} \in I(P^n, s)$.

By definition of the individuals in each $U'([s])$ we can prove that the interpretation I' is well-defined and independent from the particular choice of representatives for the equivalence classes in $U'([s])$.

Hereafter we introduce the satisfaction relation for the abstract model.

Definition 4.19*:* The satisfaction relation $\models$ for $\phi \in \mathcal{L}'$, $[s] \in \mathcal{M}'$, and an assignment σ is defined as in Def. 4.9, but for the following clauses:

$$\begin{array}{lll} (\mathcal{M}', [s], \sigma) \models P^k(\vec{t}) & \text{iff} & \langle [I'^{\sigma}(t_1)]_s, \ldots, [I'^{\sigma}(t_k)]_s \rangle \in I'(P^k, [s]) \\ (\mathcal{M}', [s], \sigma) \models \forall x \psi & \text{iff} & \text{for all } a \in U([s]),\ (\mathcal{M}', [s], \sigma^x_a) \models \psi \end{array}$$

Now, we can prove the following result on $\mathcal{M}'$.

Lemma 4. *The abstract model $\mathcal{M}'$ simulates$^+$ $\mathcal{P}$.*

The next result is immediate from Lemmas 2 and 4.

Lemma 5. *For every FO-ACTLK formula ϕ in $\mathcal{L}'$, if $\mathcal{M}' \models \phi$ then $\mathcal{P} \models \phi$.*

Thus, we have obtained an effective way of constructing the quotient system $\mathcal{M}'$ starting from $\mathcal{P}$ and $\mathcal{L}'$. More in detail, given a FO-$\forall$ACTLK formula ϕ to be model checked on $\mathcal{P}$, $\mathcal{L}'$ can be thought of as the sub-language of $\mathcal{L}$ consisting of all predicate letters and constants in ϕ.

Finally, we observe that if a QIS $\mathcal{P}$ has infinitely many states and individuals, then also the abstract model $\mathcal{M}'$ will in general be infinite both in states and individuals. However, the construction above does in some cases generate finite approximations.

Lemma 6. *Given a QIS $\mathcal{P}$, its abstract model $\mathcal{M}'$ has the following cardinality:*

$\mathcal{P}$		$\mathcal{M}'$	
$\mathcal{S}$	U	$\mathcal{S}'$	U'
infinite	*infinite*	*infinite*	*infinite*
finite	*infinite*	*finite*	*infinite*
infinite	*finite*	*finite*	*finite*
finite	*finite*	*finite*	*finite*

The proof is immediate by definition of $\mathcal{M}'$ from $\mathcal{P}$.

We have therefore identified a non-trivial case (infinite $\mathcal{S}$ and finite U) in which the technique presented above generates a feasible model checking problem. Additionally, observe that in the case of finite $\mathcal{S}$ and infinite U if all state interpretations are finite (e.g., I maps states into database instances) we also obtain, as result of the abstraction process, a finite U' in M'.

4.5 Abstraction of AC-MAS

In this section we show that under appropriate assumptions the model checking problem for FO-CTLK is decidable. We do so by proving that instead of checking the concrete, infinite-state AC-MAS $\mathcal{P}$, we can perform the verification step on an abstract, finite-state $\mathcal{P}'$, that can effectively be derived from $\mathcal{P}$, and that is equivalent to $\mathcal{P}$ w.r.t. satisfaction of FO-CTLK formulas. We start by defining a general notion of bisimilarity, i.e., a relation which captures the fact that two AC-MAS satisfy exactly the same FO-CTLK formulas. With such a notion at hand, we then introduce finite abstractions and prove that they are bisimilar to the concrete system. In the rest of the section, we let $\mathcal{P} = \langle \mathcal{S}, U, s_0, \tau \rangle$ and $\mathcal{P}' = \langle \mathcal{S}', U', s'_0, \tau \rangle$ be two AC-MAS, and assume, unless stated differently, that $s = \langle l_0, \ldots, l_n \rangle \in \mathcal{S}$ and $s' = \langle l'_0, \ldots, l'_n \rangle \in \mathcal{S}$.

4.5.1 Bisimulation

To introduce the notion of *bisimilarity* between AC-MAS [BLP12], we first need to state when two AC-MAS states are *isomorphic*.

Definition 4.20 (Isomorphism)*:* Two states s and s' are *isomorphic*, written $s \simeq s'$, iff there exists a bijection $\iota : adom(s) \cup C \mapsto adom(s') \cup C$ such that:

(i) ι is the identity on C;

(ii) for every $P_i \in \mathcal{D}$, $\vec{u} \in U^{q_i}$, and $j \in Ag$, $\vec{u} \in l_j(P_i)$ iff $\iota(\vec{u}) \in l'_j(P_i)$.

Notice that isomorphisms preserve the constants in C and the structure of the interpretation of the relation symbols in $\mathcal{D}$. In particular, it can be seen that since ι is a bijection, D_s and $D_{s'}$ are isomorphic according to the standard notion of isomorphism between relational structures. 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 state in $\mathcal{S}'$ obtained from s by renaming each $u \in adom(s)$ as $f(u)$. If f is also injective (thus invertible) and the identity on C, then $f(s) \simeq s$.

Being isomorphic is not a sufficient condition for two states to satisfy the same FO-formulas, as assignments to free variables need to be taken into account, as well. To this end, we introduce the following notion.

Definition 4.21 (Equivalent assignments)*:* Let $V \subseteq Var$ be a set of variables, and assume $s \simeq s'$. Two assignments $\sigma : Var \mapsto U$ and $\sigma' : Var \mapsto U'$ are *equivalent for* V iff there exists a bijection $\gamma : adom(s) \cup C \cup \sigma(V) \mapsto adom(s') \cup C \cup \sigma'(V)$ s.t.

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

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

Intuitively, equivalent assignments preserve all (in)equalities of variables in V, and are consistent with some witness for $s \simeq s'$. We say that two assignments are equivalent for an FO-CTLK formula φ if they are equivalent for *free*(φ).

We remark that isomorphic instances preserve non-modal first-order formulas:

Proposition 4.22*:* Given two isomorphic states s and s', an FO-formula φ, and two assignments σ and σ' equivalent for φ, we have that

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

Notice that, if φ is an FO-sentence, then it is easy to check that for all assignments σ, σ' (not necessarily equivalent), $(D_s, \sigma) \models \varphi$ iff $(D_{s'}, \sigma') \models \varphi$.

According to Prop. 4.22 isomorphic states do not distinguish FO-formulas. We now extend this result to FO-CTLK.

Definition 4.23 (Similarity)*:* The AC-MAS $\mathcal{P}'$ *simulates* $\mathcal{P}$, written $\mathcal{P} \preceq \mathcal{P}'$, iff there exists a *simulation relation* $\preceq$ on $\mathcal{S} \times \mathcal{S}'$ s.t. (i) $s_0 \preceq s'_0$; and (ii) if $s \preceq s'$ then

1. $s \simeq s'$;
2. for every $t \in \mathcal{S}$, if $s \rightarrow t$ then there is t' s.t. $s' \rightarrow t'$, $s \oplus t \simeq s' \oplus t'$ and $t \preceq t'$;
3. for every $t \in \mathcal{S}$, if $s \sim_i t$ then there is t' s.t. $t \sim_i t'$, $s \oplus t \simeq s' \oplus t'$ and $t \preceq t'$.

Observe that $\preceq$ is well-defined as $s \oplus t \simeq s' \oplus t'$ implies $t \simeq t'$. Based on the notion of simulation, can now define when two AC-MAS are *bisimilar*.

Definition 4.24 (Bisimilarity)*:* The AC-MAS $\mathcal{P}$ and $\mathcal{P}'$ are *bisimilar*, written $\mathcal{P} \approx \mathcal{P}'$, iff there exists a *bisimulation relation* $\approx$ on $\mathcal{S} \times \mathcal{S}'$ s.t. both $\approx$ and $\approx^{-1} = \{\langle s', s\rangle \mid s \approx s'\}$ are simulation relations.

Obviously, if $s \approx s'$ then $s \preceq s'$ and $s' \preceq s$. However, the viceversa is not true in general. Notice that $\approx$ is an equivalence relation.

We can prove the following result about specifications in sentence-atomic FO-CTLK.

Theorem 3. *Consider two bisimilar AC-MAS $\mathcal{P}$ and $\mathcal{P}'$, and an saFO-CTLK formula φ. Then*

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

In the next section we show how to extend this result to FO-CTLK.

4.5.2 Uniform systems

We introduce the class of AC-MAS of interest, for which we prove invariance w.r.t. FO-CTLK formulas, and the main abstraction results.

Definition 4.25 (Uniformity)*:* An AC-MAS $\mathcal{P}$ is *uniform* iff for every $s, t, s' \in \mathcal{S}$, $t' \in \mathcal{D}(U)$,

1. if $t \in \tau(s, \alpha(\vec{u}))$ and $s \oplus t \simeq s' \oplus t'$ for some witness ι, then for every bijection ι' extending ι, $t' \in \tau(s', \alpha(\iota'(\vec{u})))$.
2. if $s \sim_i t$ and $s \oplus t \simeq s' \oplus t'$, then $s' \sim_i t'$.

Intuitively, uniformity requires that transitions and epistemic relation do not depend on the data content of each $\mathcal{D}$-instance, apart from constants in C. Put differently, the requirements above capture the fact that the system is unable to distinguish among states containing the same constants and having the same data structure. This condition still allows for a vast number of AC-MAS, as it includes most systems of interest, such as those in Section 6 (but see also Section 5).

Proposition 4.26*:* If an AC-MAS $\mathcal{P}$ satisfies req. 1 in Def. 4.25 and $adom(s_0) \subseteq C$, then req. 2 is also satisfied.

We now show that uniformity, together with bisimilarity and boundedness, is sufficient to preserve FO-CTLK formulas, *Proposition* 4.27*:* If an AC-MAS $\mathcal{P}$ is uniform, then for every $s, s' \in \mathcal{S}$, $s \simeq s'$ implies $s \approx s'$.

We now prove some partial results, which are necessary to the proof of the preservation theorem for bisimilar and uniform AC-MAS.

Proposition 4.28*:* Consider two bisimilar and uniform AC-MAS $\mathcal{P}$ and $\mathcal{P}'$, two isomorphic states $s \in \mathcal{P}$ and $s' \in \mathcal{P}'$, 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, if $s \rightarrow t$ and $|U'| \geq |adom(s) \cup adom(t) \cup C \cup \sigma(free(\varphi))|$, then there exists t' s.t. $s' \rightarrow t'$, $t \approx t'$, and σ and σ' are equivalent for φ w.r.t. t and t'.
2. for every t, if $s \sim_i t$ and $|U'| \geq |adom(s) \cup adom(t) \cup C \cup \sigma(free(\varphi))|$, then there exists t' s.t. $s' \sim_i t'$, $t \approx t'$, and σ and σ' are equivalent for φ w.r.t. t and t'.

In what follows a *temporal epistemic run* r from $s \in \mathcal{S}$ is an infinite sequence $s^0 \rightsquigarrow s^1 \rightsquigarrow \dots$ such that $s^i \rightarrow s^{i+1}$ or $s^i \sim_k s^{i+1}$, for some $k \in Ag$. For $n \in \mathbb{N}$, $r(n) \doteq s^n$. A state s' is said to be *t.e. reachable from* s iff there exists a temporal epistemic run r from the initial global state $r(0) = s$ s.t. $r(i) = s'$, for some $i \geq 0$. T.e. runs are not to be confused with the (purely temporal) runs introduced in Sec. 4.1.

Prop. 4.28 generalizes to temporal epistemic runs.

Proposition 4.29*:* Consider two bisimilar uniform AC-MAS $\mathcal{P}$ and $\mathcal{P}'$, two isomorphic states $s \in \mathcal{P}$ and $s' \in \mathcal{P}'$, 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, if $r(0) = s$ and for all $i \geq 0$, $|U'| \geq |adom(r(i)) \cup adom(r(i+1)) \cup C \cup \sigma(free(\varphi))|$, then there exists a t.e. run r' 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 $r'(i)$.

We can now prove the following key result.

Lemma 7. *Consider two bisimilar and uniform AC-MAS $\mathcal{P}$ and $\mathcal{P}'$, two isomorphic states $s \in \mathcal{P}$ and $s' \in \mathcal{P}'$, 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 $|U'| \geq |adom(r(k)) \cup adom(r(k+1)) \cup C \cup \sigma(free(\varphi))| + |vars(\varphi) \setminus free(\varphi)|$;*
2. *for every t.e. run r' s.t. $r'(0) = s'$, for all $k \geq 0$ we have $|U| \geq |adom(r'(k)) \cup adom(r'(k+1)) \cup C \cup \sigma'(free(\varphi))| + |vars(\varphi) \setminus free(\varphi)|$;*

then

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

Finally, we prove the main result of this section.

Theorem 4. *Consider two bisimilar and uniform AC-MAS $\mathcal{P}$ and $\mathcal{P}'$, and an FO-CTLK formula φ. If*

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

then

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

In this section we proved that bisimilar and uniform AC-MAS satisfying assumptions 1 and 2 in Lemma 4 validate the same FO-CTLK formulas. In the next section we make use of this result to reduce, under some additional assumptions, the verification of an infinite-state AC-MAS to that of a finite-state one.

4.5.3 Finite abstractions

We now define a notion of finite abstraction for AC-MAS, and prove that it is bisimilar to the concrete model we start with. In the rest of the section we assume without loss of generality that for any AC-MAS $\mathcal{P}$, $adom(s_0) \subseteq C$. If this is not the case, C can be extended so as to include all the (finitely many) elements in $adom(s_0)$.

Definition 4.30 (Bounded AC-MAS)*:* An AC-MAS $\mathcal{P}$ is *b-bounded*, for $b \in \mathbb{N}$, if for all $s \in \mathcal{S}$, $|adom(s)| \leq b$.

Observe that boundedness imposes no restriction on the domain U of $\mathcal{P}$. Thus, even though each state in $\mathcal{P}$ does not contain more than b distinct elements, if U is infinite, so is the state space of $\mathcal{P}$ in general. The next results show that, although infinite-state, an uniform b-bounded AC-MAS $\mathcal{P}$ can in principle be verified by techniques for finite-state model checking applied to a particular AC-MAS that is a the *finite abstraction* of $\mathcal{P}$. In what follows $N_{Ag} = \sum_{A_i \in Ag} \max_{\alpha(\vec{x}) \in Act_i} \{|\vec{x}|\}$.

Definition 4.31*:* Let Ag be a set of agents, and let U' be a set. For each agent $A_i = \langle \mathcal{D}, L, Act, Pr, \tau \rangle$ in Ag we define an agent $A'_i = \langle \mathcal{D}', L', Act', Pr', \tau' \rangle$ s.t.

- $\mathcal{D}' = \mathcal{D}$;
- $L' = \mathcal{D}'(U')$;
- $Act' = Act$;
- $\alpha(\vec{u}) \in Pr'(l')$ iff there is $l \in L$ s.t. $l' \simeq l$ for some witness ι, and $\alpha(\iota'(\vec{u})) \in Pr(l)$ for some bijection ι' extending ι;

Let Ag' be the set of all A'_i.

We remark that each A'_i is indeed an agent. Notice that the definition of A'_i depends on the set U'. However, we omit U' when it is clear from the context.

Definition 4.32 (Abstraction)*:* Let $\mathcal{P}$ be an AC-MAS over Ag, the *abstraction* $\mathcal{P}' = \langle \mathcal{S}', U', s'_0, \tau' \rangle$ over Ag' is s.t.

- $s'_0 = s_0$;
- $\mathcal{S}' = \mathcal{D}(U')$ is the set of reachable states.

- $t' \in \tau'(s', \alpha(\vec{u}))$ iff there are $s, t \in \mathcal{S}$, and $\vec{u}' \in U^n$ s.t. $t \in \tau(s, \alpha(\vec{u}'))$, $s \oplus t \simeq s' \oplus t'$ for some witness ι, and $\vec{u} = \iota'(\vec{u}')$ for some bijection ι' extending ι.

Notice that $\mathcal{P}'$ is an AC-MAS. In particular, $\mathcal{P}'$ satisfies the conditions on protocols and transitions. In fact, if $t' \in \tau'(s', \alpha(\vec{u}))$, then there are $s, t \in \mathcal{S}$, and $\vec{u}' \in U^n$ s.t. $t \in \tau(s, \alpha(\vec{u}'))$, $s \oplus t \simeq s' \oplus t'$ 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 $i \leq n$. Further, $\mathcal{P}'$ is finite whenever U' is.

Lemma 8. *The abstraction $\mathcal{P}'$ of a uniform AC-MAS $\mathcal{P}$ is uniform.*

We can now prove the main result of this section.

Theorem 5. *Given an infinite, b-bounded and uniform AC-MAS $\mathcal{P}$, an FO-CTLK formula φ, and a finite set $U' \supseteq C$ s.t. $|U'| \geq 2b + |C| + \max\{vars(\varphi), N_{Ag}\}$, the abstraction $\mathcal{P}'$ is finite, uniform and bisimilar to $\mathcal{P}$. Morever,*

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

This result states that by using a sufficient number of abstract values in $\mathcal{P}'$, U', we can reduce the verification of an infinite AC-MAS to the verification of a finite one.

Further, if we drop the assumption of uniformity on $\mathcal{P}$, we still obtain a preservation result for the fragment saFO-CTLK.

Theorem 6. *Given an infinite, b-bounded AC-MAS $\mathcal{P}$, an saFO-CTLK formula φ, and a finite set $U' \supseteq C$ s.t. $|U'| \geq 2b + |C| + \max\{vars(\varphi), N_{Ag}\}$, the abstraction $\mathcal{P}'$ is finite and bisimilar to $\mathcal{P}$. Morever,*

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

Notice that U' can be taken to be any finite subset of U satisfying the condition on cardinality. 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 proper finite subsystem, which can be effectively generated.

4.5.4 The complexity of model checking FO-CTLK

We now briefly analyse the complexity of the model checking problem for finite AC-MAS w.r.t. the specification language FO-CTLK. This is tantamount to, given an AC-MAS $\mathcal{P}$ on a finite domain U and an FO-CTLK formula φ, finding an assigment σ such that $(\mathcal{P}, s_0, \sigma) \models \varphi$. Hereafter we follow [Gro01] for the setting of our investigation. We encode AC-MAS by listing the elements in the domain U, the states in $\mathcal{S}$, and all the tuples of all relations. The length of the encoding of the AC-MAS $\mathcal{P}$ is denoted by $||\mathcal{P}||$. For a database schema $\mathcal{D}$ we have $||\mathcal{P}|| = \Theta(|U| + |\mathcal{S}| + \sum_{P_i \in \mathcal{D}, D \in \mathcal{S}} |D(P_i)|)$, where $f(n) = \Theta(g(n))$ means that there exist $n_0, k_1, k_2 \in \mathbb{N}$ such that $k_1 \cdot g(n) \leq f(n) \leq k_2 \cdot g(n)$ for all $n \geq n_0$. Further, the length of the encoding of φ is denoted by $||\varphi||$. We now state the following complexity result.

Theorem 7. *The complexity of the model checking problem for finite AC-MAS w.r.t. the language FO-CTLK is PSPACE-complete.*

In this section we studied the model checking problem for AC-MAS. Even if it is of interest to us, we do not consider here the implicit model checking problem [Sch02] defined directly on AS programs. We do remark though that Theorem 7 is nonetheless a notable result as it shows that for AC-MAS the complexity of model checking FO-CTLK formulas is better than the complexity of checking the propositionalisation φ' of a FO-CTLK formula φ, as φ' is usually exponential in the size of φ [HVCG07].

5 Verification of GSM-based Artifact Systems

In this section we take a step towards implementations and instantiate the AC-MAS semantics by means of artifact system descriptions that are close to what used in practice. We show that the setting complies with the conditions explored in the previous section thereby enabling us to show results for artifact-centric implementations. By doing so we achieve two results. Firstly, we provide a formal semantics to GSM programs via AC-MAS that can be used to interpret FO-CTLK specifications. Secondly, this enables us to apply the finite abstraction methodology in Section 4.1 to GSM programs. The material in this section has not appear in D2.1.1 and has been submitted to a major conference.

5.1 GSM Programs

Artifact-centric systems are based on the notion of *artifact*, or a record of structured data, that are born, evolve, and die during a system run either as a consequence of chains of internal actions of other artifacts, or through external actions performed by actors. GSM [ea11] is a declarative language, interpreted by specialised toolkits, that enables the user to implement sophisticated guard-stage-milestone models for artifact systems.

For simplicity, here we work on an untyped version of GSM programs in which we also neglect timestamps. While GSM programs are richer, the version we consider enables us to present concisely our decidability results while at the same time supporting complex usecases as we show in Section 5.5.

Definition 5.1 (Artifact Type)*:* An *artifact type* is a tuple $AT = \langle P, x, Att, Stg, Mst, Lcyc \rangle$ s.t.

- P is the name of the artifact type;
- x is a variable that ranges over the IDs of instances of AT;
- Att is the set of attributes, which is partitioned into the set Att_{data} of data attributes and Att_{status} of status attributes;
- Stg is the set of stages;
- Mst is the set of milestone;
- $Lcyc$ is the lifecycle model of the artifact type AT.

Att_{data} includes the attribute $mostRecEventType$, which holds the type of the most recent event. For each milestone $m \in Mst$, in Att_{status} there is a boolean *milestone status value* attribute, denoted as m. Analogously, for each stage $S \in Stg$, in Att_{status} there is a boolean *stage status value* attribute, denoted as $active_S$.

While the data content of an artifact type is specified by its datamodel, i.e., all its attributes excluding its lifecycle, its execution is described by its lifecycle.

Definition 5.2 (Lifecycle)*:* The lifecycle of an artifact type AT is a tuple $Lcyc = \langle Substg, Task, Owns, Guards, Ach, Inv \rangle$ s.t.

- $Substg$ is a function from Stg to finite subsets of Stg, where the relation $\{(S, S') | S' \in Substg(S)\}$ is a forest. The leaves are *atomic* stages.

- $Task$ is a function from the atomic stages in Stg to tasks.
- $Owns$ is a function from Stg to finite, non-empty subsets of Mst, such that $Owns(S) \cap Owns(S') = \emptyset$ for $S \neq S'$.
- $Guards$ is a function from Stg to finite sets of sentries.
- Ach is a function from Mst to finite sets of *achieving* sentries.
- Inv is a function from Mst to finite sets of *invalidating* sentries.

Intuitively, every artifact goes through a number of stages, which are activated by the relevant guards. A stage is closed when the tasks associated with it and its substages are fulfilled. When this happens the milestones associated with the stage become true and possibly trigger the guards associated with another stage.

We now introduce GSM programs.

Definition 5.3 (GSM program)*:* A GSM program Γ is a set of artifact types AT_i for $i \leq n$.

As a convenience, we also assume that all of the context variables are distinct.

Artifact instances are then defined as mappings from artifact types to a possibly infinite interpretation domain U.

Definition 5.4 (AI and GSM snapshot)*:* A *snapshot* for the artifact type AT is a mapping σ from x, Att to the domain U.

A *snapshot* for the GSM program Γ is a mapping Σ from each type $AT \in \Gamma$ to a set $\Sigma(AT)$ of snapshots for type AT.

Notice that different instances of the same artifact type are distinguished by their IDs $\sigma(x)$, usually denoted as ρ.

We assume that no confusion will arise from the use of σ for assignments in AC-MAS and snapshots in GSM programs. The context will disambiguate. In [ea11] snapshots are assumed to satisfy the extra conditions **GSM1-3**, which we omit. In fact, it can be shown that **GSM1-3** always hold by definition of execution for GSM programs.

5.2 Execution of GSM programs

In GSM programs events are responsible for the evolution of the system from one snapshot to the next. Three kinds of incoming events are considered: 1-way messages M, 2-way service call returns F^{return}, and artifact instance creation requests $create^{call}_{AT}$. A ground event e has a *payload* $(A_1 : c_1, \ldots, A_n : c_n)$ where A_i is a data attribute and c_i is a value in the interpretation domain U. Intuitively, incoming events are processed by the sentries associated with guards and milestones.

Definition 5.5 (Immediate effect)*:* The *immediate effect* of a ground event e on a snapshot Σ, or *ImmEffect*(Σ, e), is the snapshot that results from incorporating e into Σ, including

- changing the values of the *mostRecEventType* attribute of affected (or created) artifact instances;
- changing the values of data attributes of affected artifact instances as indicated by the payload of e.

The operational semantics for GSM programs is built around the notion of *business step*, or B-step, which represents the impact of a single ground incoming event e on a snapshot Σ. The semantics of B-steps is characterised by 3-tuples (Σ, e, Σ') s.t.

1. Σ is the *previous* snapshot;
2. e is a ground incoming event;
3. Σ' is the *next* snapshot;
4. there is a sequence $\Sigma_0, \Sigma_1, \ldots, \Sigma_n$ of snapshots s.t. (i) $\Sigma_0 = \Sigma$; (ii) $\Sigma_1 = \mathit{ImmEffect}(\Sigma, e)$; (iii) $\Sigma_n = \Sigma'$; and (iv) for $1 \leq j \leq n-1$, Σ_{j+1} is obtained from Σ_j by a PAC rule.

The semantics of GSM programs is Business steps are based on a variation of Event-Condition-Action rules, called *Prerequisite-Antecedent-Consequent* (PAC) rules. To introduce PAC rule we first define formally event expressions and sentries. In what follows τ_{AT} is a path expression $x. < path >$ where x is the ID variable for $AT \in \Gamma$. We assume that no confusion arises with the transition function τ for AC-MAS.

Definition 5.6 (Event expression)*:* An *event expression* $\xi(x)$ for an artifact type AT with ID variable x has one of the following forms:

- **Incoming event expression** $x.e$**:** (i) $x.M$ for 1-way message type M; (ii) $x.F^{return}$ for service call return from F; (iii) $x.create_{AT}^{call}$ for a call to create an artifact instance of type AT.
- **Internal event expression:** (i) $+\tau_{AT'}.m$ and $-\tau_{AT'}.m$, where m is a milestone for type AT'; (ii) $+\tau_{AT'}.active_S$ and $-\tau_{AT'}.active_S$, where S is a stage of type AT'.

We can now define sentries for guards and milestones. These represent the conditions to open and close stages.

Definition 5.7 (Sentry)*:* A *sentry* for an artifact type AT is an expression $\chi(x)$ having one of the following forms: **on** $\xi(x)$ **if** $\varphi(x) \wedge x.active_S$, **on** $\xi(x)$, or **if** $\varphi(x) \wedge x.active_S$ s.t. (i) $\xi(x)$ is an event expression; and (ii) $\varphi(x)$ is an FO-formula over the artifact types occurring in Γ that has exactly one free variable.

We now briefly discuss when a snapshot Σ satisfies a sentry χ, or $\Sigma \models \chi$. Satisfaction of an FO-formula φ at Σ is straightforward. Further, the expression $\rho.e$ for an artifact instance ρ is true at Σ if $\rho.mostRecEventType = e$. Finally, the internal event expression $\odot\rho.\tau.s$ for polarity $\odot \in \{+, -\}$, path expression τ, and status attribute s, is true at Σ if the value of $\rho.\tau.s$ matches the polarity.

Definition 5.8 (PAC rules)*:* A PAC rule is a tuple $\langle \pi(x), \alpha(x), \gamma(x) \rangle$ s.t.

- $\pi(x)$ is of the form $\tau.m$, $\neg\tau.m$, $\tau.active_S$, or $\neg\tau.active_S$;
- $\alpha(x)$ is of the form $\chi(x) \wedge \psi(x)$, where $\chi(x)$ is a sentry and $\psi(x)$ is of the form $\tau.m$, $\neg\tau.m$, $\tau.active_S$, or $\neg\tau.active_S$;
- $\gamma(x)$ is an internal event expression as in Def. 5.6.

Given a B-step $\Sigma = \Sigma_0, \Sigma_1, \ldots, \Sigma_j$ for a ground event e, the PAC rule $\langle \pi, \alpha, \gamma \rangle$ is applicable if $\Sigma \models \pi$ and $\Sigma_j \models \alpha$. Applying such a rule yields a new snapshot Σ_{j+1}, which is constructed from Σ_j by applying the effect called for by γ.

Additional conditions are also assumed for the application of PAC rules, which notably ensure the absence of cycles. We do not present these here and refer to [ea11] for further information.

5.3 Embedding of GSM Programs into AC-MAS

In this section we introduce an embedding of GSM programs into AC-MAS. By doing so we achieve two results. Firstly, we provide a formal semantics to GSM programs via AC-MAS that can be used to interpret FO-CTLK specifications. Secondly, this enables us to apply the finite abstraction methodology in Section 4.1 to GSM programs.

To begin, for each artifact type $AT = \langle P, x, Att, Stg, Mst, Lcyc\rangle$ we introduce a predicate symbol P with attributes x, Att. Hence, the arity of P is $q_P = 1 + |Att|$.

Definition 5.9*:* Given a GSM program $\Gamma = \{AT_j\}_{j\leq n}$ we define a database schema $\mathcal{D}_\Gamma = \{P_j\}_{j\leq n}$ s.t. each P_j is the predicate symbol corresponding to the artifact type AT_j.

We now introduce agents in GSM programs.

Definition 5.10*:* Given a GSM program Γ and an interpretation domain U, an agent is a tuple $i = \langle \mathcal{D}_i, L_i, Act_i, Pr_i\rangle$ s.t.

- $\mathcal{D}_i \subseteq \mathcal{D}_\Gamma$ is the *local database schema*, and $\mathcal{D}_E = \mathcal{D}_\Gamma$;
- $L_i = \mathcal{D}_i(U)$ is the set of *local states*, and $L_E = \mathcal{D}_\Gamma(U)$;
- Act_i is the set of actions $\alpha_e(\vec{y})$ for each event e with formal payload $\vec{y}$. Further, we introduce a skip action $skip_i$ for each agent i. Act_E is defined similarly.
- For every ground action $\alpha_i(\vec{u})$, for every local state l_i, $\alpha_i(\vec{u}) \in Pr_i(l_i)$, i.e., a ground action $\alpha_i(\vec{u})$ is always enabled.

We observe that the original formulation of GSM programs in [ea11] does not account for agents. In fact, artifacts are bundled together in the *Interaction Hub* (IH), which interacts with the external environment through incoming and generated events. By Def. 5.10 the Interaction Hub of GSM programs is mapped into the environment of AC-MAS; while the environment of GSM programs is mapped to agents in AC-MAS. So, the notion of environment corresponds to different entities in GSM programs and AC-MAS. We keep the original terminology, as the distinction is clear. Furthermore, each agent, including the environment, perform actions corresponding to sending an event to the Interaction Hub. Actions are always enabled as no protocol is explicitly given for GSM programs.

Given a set of agents defined as above, the AC-MAS $\mathcal{P}_\Gamma$ associated to the GSM program Γ is defined according to Def. 4.7. Specifically,

Definition 5.11 (AC-MAS)*:* Given a set Ag of agents over the GSM program Γ and a snapshot Σ_0, the AC-MAS associated with Γ is a tuple $\mathcal{P}_\Gamma = \langle \mathcal{S}, U, \Sigma_0, \tau\rangle$ s.t.

- $\mathcal{S} \subseteq L_e \times L_1 \times \cdots \times L_n$ is the set of *reachable global states*;
- U is the interpretation domain;
- $D_{\Sigma_0} \in \mathcal{S}$ is the *initial global state* corresponding to Σ_0;
- $\tau : \mathcal{S} \times Act(U) \mapsto 2^{\mathcal{S}}$ is the *global transition function* s.t. $D' \in \tau(D, \alpha(\vec{u}))$ iff (i) if $\alpha = \langle \alpha_e, \alpha_1, \ldots, \alpha_n\rangle$ then at most one α_i is different from $skip_i$; (ii) if $\alpha_i = \alpha_e$ then $(\Sigma_D, e, \Sigma_{D'})$ holds in Γ, where $\vec{u}$ is the payload of event e.

Notice that there is a one-to-one correspondence between snapshots in Γ and states in the AC-MAS $\mathcal{P}_\Gamma$. Given a snapshot Σ we denote the corresponding state as D_Σ. Also, GSM programs do not specify initial states; therefore the definition of $\mathcal{P}_\Gamma$ is parametric in Σ_0, i.e., the snapshot chosen as the initial state of Γ. Most importantly, the transition function τ mirrors the B-step semantics of GSM programs. Since each B-step consumes a single event, we require that at most one agent performs an event action at each given time, while all other agents keep idle. This has correspondences with other approaches based on multi-agent systems such as interleaved interpreted systems [LPQ10].

5.4 Finite abstractions of GSM programs

In this section we show that GSM programs admit finite abstractions. Specifically, by suitably restricting the language of sentries we can prove that the AC-MAS $\mathcal{P}_\Gamma$ obtained from a GSM program Γ is uniform. So, by applying Theorem 5 we obtain that if $\mathcal{P}_\Gamma$ is also bounded, then it admits a finite abstraction, hence its model checking problem is decidable. The key notion of this subsection is that of *amenable* GSM program.

Definition 5.12 (Amenable GSM programs)*:* A sentry $\chi(x)$ is *amenable* if $\varphi(x)$ in $\chi(x)$ belongs to language $\mathcal{L}_{\mathcal{D}_\Gamma}$, i.e., $\varphi(x)$ is an FO-formula built on the predicate symbols in $\mathcal{D}_\Gamma$.

A GSM program is *amenable* iff all sentries occurring in some guards and milestones are amenable.

It is known that, given a database schema $\mathcal{D}$, the language $\mathcal{L}_\mathcal{D}$ built on it is sufficiently expressive to define a wide range of systems [BLP12, HCG$^+$11]. As an example, the scenario we investigate in the next section adheres to this property. Therefore we see amenable GSM programs as an interesting and powerful class of GSM programs with potentially wide applicability.

The next results show that the AC-MAS $\mathcal{P}_\Gamma$ is uniform whenever Γ is amenable.

Lemma 9. *For every $D, D' \in \mathcal{P}_\Gamma$, if $D \simeq D'$ for some witness ι, then $\Sigma_{D'} = \iota(\Sigma_D)$.*

The next result is key in the proof of uniformity for $\mathcal{P}_\Gamma$.

Lemma 10. *For every D, D' , $D'' \in \mathcal{S}$, $D''' \in \mathcal{D}_\Gamma(U)$, if $D \oplus D' \simeq D'' \oplus D'''$ for some witness ι, then*

$$(\Sigma_D, e, \Sigma_{D'}) \quad \textit{implies} \quad (\Sigma_{D''}, \iota'(e), \Sigma_{D'''})$$

where ι' is any bijection extending ι

This enables us to state the first of our two key results.

Theorem 8. *If the GSM program Γ is amenable, then the AC-MAS $\mathcal{P}_\Gamma$ is uniform.*

By combining Theorem 5 with 8 we obtain a decidable model checking procedure for amenable GSM programs. Specifically, a GSM program is b-bounded if the cardinality of all snapshots is bounded, i.e., there is a $b \in \mathbb{N}$ s.t. $|\Sigma| \leq b$ for all snapshots Σ. Hence, we have the following result:

Corollary 5.13*:* Assume a b-bounded GSM program Γ on an infinite domain U_1, an FO-CTLK formula φ, and a finite set $U_2 \supseteq C$ s.t. $|U_2| \geq 2b + |C| + \max\{var(\varphi), N_{Ag_1}\}$. Then, the abstraction $\mathcal{P}'$ of $\mathcal{P}_\Gamma$ is uniform and bisimilar to $\mathcal{P}_\Gamma$. Moreover, $\mathcal{P}_\Gamma \models \varphi$ iff $\mathcal{P}' \models \varphi$.

Thus, to verify a GSM program we can model check the finite abstraction of the corresponding AC-MAS. By the remarks at the end of Section 4.5.3 the latter procedure can be computed effectively.

To conclude, in Section 4.5.4 it was proved that the model checking problem for finite AC-MAS is PSPACE-complete in the size of the state space $\mathcal{S}$ and the specification φ. So, we obtain the following:

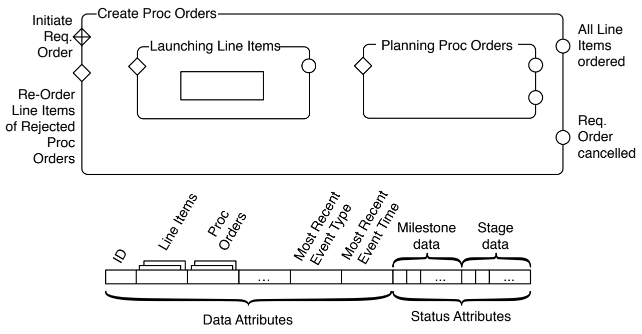


Figure 1 – The Requisition Order datamodel and part of its lifecycle.

Proposition 5.14*:* Model checking amenable GSM programs is PSPACE-hard in the number of snapshots and the length of the specification.

5.5 Case Study

The *Requisition and Procurement Orders* (RPO) scenario is a business process use-case in which a GSM program is used to implement the procurement process in a business setting [ea11]. We illustrate some of the results of the section in the context of a fragment of this scenario. In RPO a *Requisition Order* is sent by a Customer to a Manufacturer to request some goods. The Requisition Order has one or more *Line Items*, which are bundled into Procurement Orders and sent to different Suppliers. A Supplier can either accept or reject a Procurement Order. In the latter case, the rejected Line Items are bundled into new Procurement Orders.

In GSM programs the Requisition and Procurement Orders are implemented as artifact types. Specifically, the *datamodel* of the Requisition Order, i.e., all its attributes excluding the lifecycle, are encoded as in the lower part of Fig. 1. Notice that in the datamodel we have both data and status attributes; the latter contains milestone and stage data.

The upper part of Fig. 1 illustrates part of the lifecycle for the Requisition Order. Stages are represented as rounded boxes. The stage Creating Proc Orders contains the child-stages Launching Line Items and Planning Proc Orders; the former is atomic. Milestones are shown as small circles associated with stages. For instance, the milestones All Line items ordered and Req Order cancelled are associated with the stage Creating Proc Orders. The former is an achieving milestone, i.e., when All Line items ordered becomes true the stage is closed; while the latter is invalidating, that is, when Req Order cancelled holds, the stage is then reopened. The diamond nodes are guards. The stage Creating Proc Orders is triggered by the guards Initiate Req Order and Re-order Line Items of Rejected Proc Orders. A diamond with a cross represents a “bootstrapping” guard, which indicates the conditions to create new artifact instances.

As discussed in Section 5 the execution of GSM programs is governed by PAC rules. To illustrate these we consider PAC2 in [ea11]:

	Prerequisite π	Antecedent α	Consequent γ
PAC2	$x.active_S$	**on** $e(x)$ **if** $\varphi(x)$	$+x.m$

where stage S has milestone m and **on** $e(x)$ **if** $\varphi(x)$ is an achieving sentry for m. Suppose that $\Sigma_0, \Sigma_1, \ldots, \Sigma_j$ is a sequence of snapshots in a B-step. Intuitively, if $\Sigma \models \pi$ then there is an artifact instance ρ s.t. $\rho.active_S$ is true, i.e., the stage S is active for ρ. Furthermore, if $\Sigma_j \models \alpha$, then $\rho.mostRecEventType = e$ and the achieving condition φ for milestone m holds. Finally, Σ_{j+1} is obtained by applying $+\rho.m$, i.e., by toggling true the flag m for the milestone status value of S.

Now we briefly show how the GSM program *RPO* for the RPO scenario translates into the corresponding AC-MAS $\mathcal{P}_{RPO}$. Firstly, we associate the RPO scenario with the database schema $\mathcal{D}_{RPO}$ containing a predicate symbol P_{RO} for the Requisition Order artifact type, as well as a predicate symbol P_{PO} for the Procurement Order artifact type. In particular, the predicate symbol P_{RO} has data and status attributes as specified in the datamodel in Fig. 1.

A number of agents appears in the RPO scenario: a Customer C, a Manufacturer M, and one or more Suppliers S. By Def. 5.10 each agent has a partial view of the database schema $\mathcal{D}_{RPO} = \{P_{RO}, P_{PO}\}$. We can assume that the Customer can only access the Requisition Order (i.e. $\mathcal{D}_{\mathsf{C}} = \{P_{RO}\}$), and the Supplier only the Procurement Order (i.e. $\mathcal{D}_{\mathsf{S}} = \{P_{PO}\}$), while the Manufacturer can access both (i.e. $\mathcal{D}_{\mathsf{M}} = \{P_{RO}, P_{PO}\} = \mathcal{D}_{RPO}$). Finally, the AC-MAS $\mathcal{P}_{RPO}$ defined according to Def. 5.11, is designed to mimic the behaviour of the GSM program *RPO* for the RPO scenario. In particular, we have a temporal transition $D \rightarrow D'$ in $\mathcal{P}_{RPO}$ iff there is some event e s.t. $\langle \Sigma_D, e, \Sigma_{D'} \rangle$ holds in *RPO*.

By means of the AC-MAS $\mathcal{P}_{RPO}$ we can model check *RPO* against first-order temporal epistemic specifications. For instance, the following FO-CTLK formula specifies that the manufacturer M knows that each Procurement Order has to match a corresponding Requisition Order:

$$\phi \;=\; AG\, \forall ro\_id, \vec{x}\ (PO(id, ro\_id, \vec{x}) \rightarrow K_{\mathsf{M}}\ \exists \vec{y}\ RO(ro\_id, \vec{y}))$$

We remark that the GSM program *RPO* can be defined so that any clause $\varphi(x)$ in some sentry $\chi(x)$ belongs to the FO-language $\mathcal{L}_{\mathcal{D}_{RPO}}$. Hence, *RPO* is amenable. Finally, by Def. 4.32 we can introduce the finite abstraction $\mathcal{P}'$ of $\mathcal{P}_{RPO}$ as the subsystem defined on a finite subset of the interpretation domain satisfying the cardinality condition. By Corollary 5.13 we can check whether *RPO* satisfies ϕ by model checking it in the finite abstraction $\mathcal{P}'$.

5.6 Discussion

This section stems from an observation regarding the current lack of support for verification in GSM environments. While abstraction methodologies for various artifact-inspired systems have been put forward [HCG+11, BLP12, DHPV09], they all lack support for program verification and operate on logical models, thereby making automatic model checking impracticable. Our objective in this section was to surpass this severe limitation and position the GSM approach firmly in an agent-based semantics so that information-theoretic properties such as knowledge of the participants could be verified. We achieved this by extending minimally the semantics of AC-MAS [BLP12] to account for parametric actions while at the same time maintaining the key results concerning finite abstractions. We then proceeded to map constructs of GSM into the corresponding constructs of the revised AC-MAS and identified what we called "amenable" GSM programs that we showed to admit finite abstractions. We remarked that amenable GSM is not a significant limitation in applications and showed the key passages of the approach presented on a fraction of a notable usecase from [ea11]. The work in this paper brings us considerably closer to an implementation for automatic model checking of GSM programs.

6 Foundations of Relational Artifacts Verification

In this section we study the foundations of artifact-centric systems that use relational databases for their data component. Specifically, we consider several artifacts (fixed in advance) forming a *relational artifact system*, each constituted by a *relational database* evolving over time. To characterize such an evolution, we rely on a very rich notion of lifecycle, directly based on stating dynamic properties in terms of *intra-artifact* and *inter-artifact dynamic constraints*. We express such constraints, and other dynamic properties of interest, in a suitable variant of μ-calculus, one of the most expressive temporal logics used in verification [LPP70, Eme96].

We consider *processes* over artifacts constituted by a set of *actions* (aka atomic tasks, atomic services) and a set of *condition-action rules*, which specify when such actions can be executed. The action specification is possibly the most characterizing part of our framework. Following [CGRR10], actions are specified in terms of preconditions and postconditions on artifacts' databases. Such a specification is strongly influenced by the notion of *mappings* in the recent literature on data exchange and data integration [FKMP05, Len02]. In a nutshell, our action specification considers the current state of the database, and the one obtained by executing an action as two databases related through a set of mappings. In the literature, mappings typically establish correspondences between conjunctive queries, also called tuple-generating dependencies (TGDs) in the database jargon [AHV95b]. However here, differently from [CGRR10], we do use negation and more generally full first-order queries in defining the preconditions of actions. Technically speaking, this choice requires us to abandon the theory of conjunctive queries and homomorphisms at the base of the results in [CGRR10, FKMP05, Len02].

We are interested in two main reasoning tasks. The first one is *conformance of a process to an artifact system*, which consists in checking whether a given process generates the correct lifecycle for the various artifacts and, more generally, whether it satisfies all intra-artifact and inter-artifact constraints. The second reasoning task is *process verification*, that is checking whether a process (over an artifact system) verifies general dynamic properties of interest. Both these reasoning tasks in principle can be based on model checking, though, in our setting, one has to deal with potentially infinite states.

We show that both reasoning tasks are undecidable even for very simple artifact systems and processes. We then introduce a very interesting class of processes for which decidability is granted. We call such processes *weakly acyclic*, since they satisfy a condition analogous to weak acyclicity of a set of mappings in data exchange [FKMP05]. Under such a restriction, we are guaranteed that the number of new objects introduced by the execution of actions is finite, and hence, the whole process is finite-state.

The rest of the section is organized as follows. Sections 6.1, 6.2, and 6.3 introduce the framework and illustrate it through a running example. Section 6.4 reports the undecidability of conformance and verification in our framework even for simple cases. Section 6.5 introduces the notion of weakly acyclic processes, and shows that such a restriction makes both conformance and verification decidable. Section 6.6 discusses briefly further works.

The material in this work has appeared in D2.1.1 and has been published in [HCG+11].

6.1 Relational Artifacts Systems

In this section, we start the description of our framework by introducing relational artifact systems. In the following, we assume the reader to be familiar with standard relational databases, and their connection with first-order logic (FOL). In particular, queries are seen as (possibly open) FOL formulas. Also, we consider as special FOL queries conjunctive queries (CQs), i.e., formulas formed only by conjunctions and existential quantifications, and their unions (UCQs) [AHV95b].

A *relational artifact systems* RAS is constituted by a set of artifacts, each formed by a relational database evolving over time under restrictions imposed by certain dynamic constraints. We deal with two types of constraints: the *intra-artifact dynamic constraints*, that involve each artifact in isolation, and the *inter-artifact dynamic constraints*, taking into account relations between artifacts. In this section we introduce such systems.

6.1.1 Relational artifact

A relational artifact is a relational database evolving over time. Hence, it is characterized by the usual notions of *database schema*, giving the structure of the database, and *database instance*, detailing the actual data contained in it, and it is furthermore augmented by a set of *intra-artifact dynamic constrains*. These are temporal constraints expressed in the temporal logic $\mu\mathcal{L}$ introduced later, which allows us to express various constraints over the database: we can assert the usual ones, such as inclusion dependencies, which now become safety temporal constrains, and also what is typically called the *artifact lifecycle*, namely, dynamic constrains on the sequencing of configurations the database may pass through. More formally, a *relational artifact* is a tuple $A = \langle \boldsymbol{R}, I_0, \Phi \rangle$ where:

- $\boldsymbol{R} = \{R_1, \ldots, R_n\}$ is a database schema, constituted by a set of relation schemas;
- I_0 is a database instance, compliant with the schema $\boldsymbol{R}$, that represents the initial state of the artifact;
- Φ is a $\mu\mathcal{L}$ formula over $\boldsymbol{R}$ constituted by the conjunction of all intra-artifact dynamic constraints of A.

Notice that if we project the dynamic formula Φ over the initial artifact instance I_0, we may get (depending on the structure of Φ) static, i.e., local, constraints on I_0. From now on, we assume to deal with *well formed artifacts*, namely, artifacts whose initial instance satisfies such local constraints.

6.1.2 Relational artifact system

A relational artifact system is composed by several relational artifacts in execution at the same time, each consisting of a database and a set of intra-artifact dynamic constraints. The dynamic interaction between them is regulated through additional constraints, also expressed in $\mu\mathcal{L}$, which we call *inter-artifact-dynamic constraints*.

In the following, we make the assumption that artifacts cannot be created or destroyed during the evolution of the system. Under it, we get quite interesting undecidability and decidability results. We are indeed very interested in dropping these limitations in future works, starting from the results presented here. For this reason we start with a finite set of artifacts, and over the whole evolution of the system these will remain the only ones of interest. If an artifact has a terminating lifecycle it becomes dormant, but it will persist in the system.

Formally, an *artifact system* is a pair $\mathcal{A} = \langle\{A_1, \ldots, A_n\}, \Phi_{inter}\rangle$, where $\{A_1, \ldots, A_n\}$ is the finite set of artifacts of the system (each with its own database and intra-artifact dynamic constraints expressed in $\mu\mathcal{L}$), and Φ_{inter} is a $\mu\mathcal{L}$ formula expressing the conjunction of inter-artifact dynamic constraints. To distinguish relations of various artifacts in $\mathcal{A}$, we use the usual *dot notation* of object-orientation, hence, a relation R_j of artifact A_i of $\mathcal{A}$ is denoted by $A_i.R_j$. When clear from the context, we drop the artifact A_i and we use R_j for the relation. We denote by $\mathcal{I}_0$ the disjoint union of all initial instances of the artifacts in $\mathcal{A}$, i.e., $\mathcal{I}_0 = \bigcup_{i=1,\ldots,n} I_{0,i}$. More generally, $\mathcal{I}$ represents the instance obtained by the (disjoint) union of the current instances of each artifact in $\mathcal{A}$.

Given a database instance I, we denote by *constI* the active domain of I, i.e., the set of individuals (typically constants) appearing in I. Hence, the active domain of $\mathcal{I}_0$ is $const\mathcal{I}_0$, which is made up by all constants appearing in the initial instances of the various artifacts in $\mathcal{A}$.

Notice that, while artifact systems evolve over time, they do not include a predefined mechanism for progression. Progression is due to the execution of actions, tasks, or services over the system, according to a given process that we will introduce later on. Here it is sufficient to assume that a progression mechanism exists, and its execution results in moving from the initial state, given by the instance $\mathcal{I}_0$, to the next one, and so on.

In this way we build a *transition system* [CGP99] $\mathfrak{A}$, whose states represent possible system instances, and each transition an atomic step in the progression mechanism (whatever it is). In principle, we can model-check such a transition system to verify dynamic properties [CGP99], that is exactly what we are going to do next. However, one has to consider that, in general, $\mathfrak{A}$ is infinite, hence the classical results on model checking [CGP99, Eme96], which are developed for finite transition systems, do not apply. The main goal of this work is to find interesting conditions under which such a transition system is finite.

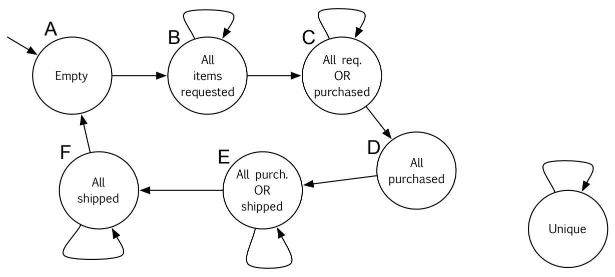
Example 1*:* We model the process of purchasing items within a company. In particular, when a company's employee, that assumes the role of a *requester*, wants to purchase some items, he has to turn to a *buyer*, also internal to the company, who is responsible for purchasing such items from external *suppliers*. In our scenario, we have five actors: two requesters (Bob and Alice), a buyer (Trudy) and two suppliers (SupplierA and SupplierB). The whole purchasing process works as follows: in a first phase, the requester has to fill a so-called *requisition order* with some *line items* chosen from a catalogue. In our simple example the catalogue contains only a monitor, a mouse and a keyboard. Once the requester has completed this process, he sends the order to the buyer, which extracts the line items from it, and purchases each of them separately. In particular, the buyer groups together into a *procurement order* line items (belonging to a requisition order) that will be purchased from a particular supplier. As a result of this phase, we get different procurement orders, each containing line items that the buyer requests from a single supplier. Then the supplier ships back to the buyer the items included in the procurement order he received, and finally, the items are delivered to the original requester. Of course, we can have many orders processed simultaneously in the system, although we will impose some restrictions.

In this example, we consider the relational artifact system $\mathcal{A} = \langle\{\mathsf{ReqOrders}, \mathsf{ProcOrders}\}, \Phi_{inter}\rangle$ containing two relational artifacts, holding all relevant data about requisition orders and procurement orders in the system.

$\mathsf{ReqOrders} = \langle \boldsymbol{R}_{RO}, I_{0,RO}, \Phi_{RO}\rangle$

- $\boldsymbol{R}_{RO} = \{\ \mathsf{RO}(RoCode, ReqName, BuName),\ \mathsf{ROItem}(RoCode, ProdName, Status),$
  $\mathsf{Requester}(ReqName),\ \mathsf{LineItem}(ProdName, Price),$
  $\mathsf{Buyer}(BuName),\ \mathsf{Status}(StatusName)\ \}$

  A requisition order is meant to hold the data associated to every *pending* requisition order: indeed, as soon as the items are delivered to the corresponding requester, each information



(a) (b)

Figure 2 – Informal representation of *dynamic* intra-artifact constraints

associated to them is removed from the system. Relation $\mathsf{RO}(RoCode, ReqName, BuName)$ holds basic information associated to a single order i.e., order's code and both requester's and buyer's names. The requested items are kept in the relation $\mathsf{ROItem}(RoCode, ProdName, Status)$, whose attribute *Status* keeps track of the status of each line item included in the order (it can be either requested, purchased or shipped). Relations $\mathsf{Requester}(ReqName)$, $\mathsf{LineItem}(ProdName, Price)$, $\mathsf{Buyer}(BuName)$ and $\mathsf{Status}(StatusName)$ are included in the schema for technical convenience; in particular, the relation Status is needed in order to easily bind values of the attribute *Status* of each line item in an order.

- $I_{0,RO} = \{$ Requester(Bob), Requester(Alice), Buyer(Trudy),
  Status(requested), Status(purchased), Status(shipped),
  LineItem(keyboard, 20), LineItem(mouse, 10), LineItem(monitor, 200) $\}$

  According to the previous description of this example scenario, in the initial instance we only have data referred to existing requesters, buyers, suppliers and the catalogue, featuring three line items. There are no pending orders.

- As for intra-artifact constraints, here we only give an intuition of what will be presented formally later. We want to trace the status of an ordered line item through the attribute $\mathsf{ROItem}.Status$, so we express constraints on the evolutions of all orders in the system by relying on this attribute, as informally depicted in Figure 2a. Intuitively, at the beginning, we do not have any order placed by requesters: in the current situation (henceforth called *phase*) the relations RO and ROItem are empty $[A]$. As orders are placed, and new requisition orders are created, we will have a phase in which all currently pending orders have status requested $[B]$, and such condition will hold until, *eventually*, some item in such status will be purchased by the buyer by creating procurement orders to send to suppliers, hence changing status to purchased. At this point, we will be in a phase such that all items belonging to existing orders are either requested or purchased $[C]$ and finally, at some point, all orders will be purchased $[D]$. Having all procurement orders sent to suppliers, some of them will be shipped back, i.e., setting the status of corresponding items to shipped $[E]$, and at the end, all of them will be shipped back to the buyer $[F]$. Finally, as the items are delivered to the requester, they will be removed from the system and the initial condition will be eventually met again. Notice that we are imposing some restrictions on the evolution of the artifact: for instance, we do not allow new requisition orders (for new line items) to be created unless all the existing ones have been purchased $[D]$. Notice also that we will need a way to force the system to eventually exit self-loops. Moreover, in addition to such *dynamic* constraints, we also have some *static* ones, such as inclusion dependencies.

ProcOrders $= \langle \boldsymbol{R}_{PO}, I_{0,PO}, \Phi_{PO} \rangle$

- $\boldsymbol{R}_{PO} = \{$ $\mathsf{PO}(PoCode, RoCode, SupName)$, $\mathsf{POItem}(PoCode, RoCode, ProdName)$,
  $\mathsf{Supplier}(SupName)$, $\mathsf{LineItem}(ProdName, Price)\}$.

  Recall that all line items assigned to the same procurement order must belong to the same requisition order. Hence, similarly to requisition orders, a procurement order's schema includes a relation $\mathsf{PO}(\mathit{PoCode}, \mathit{RoCode}, \mathit{SupName})$ holding its code, the code of the corresponding requisition order and the name of the chosen supplier. Relation $\mathsf{POItem}(\mathit{PoCode}, \mathit{RoCode}, \mathit{ProdName})$ holds instead the set of line items in each procurement order. Attribute *RoCode* is replicated in this relation for convenience. $\mathsf{Supplier}(\mathit{SupName})$ keeps the set of existing suppliers whereas $\mathsf{LineItem}(\mathit{ProdName}, \mathit{Price})$ is the same as the one in requisition order artifact.

- $I_{0,PO} = \{\ \mathsf{LineItem}(\mathsf{keyboard}, 20), \mathsf{LineItem}(\mathsf{mouse}, 10), \mathsf{LineItem}(\mathsf{monitor}, 200),$
  $\mathsf{Supplier}(\mathsf{SupplierA}), \mathsf{Supplier}(\mathsf{SupplierB})\}.$
- In this example we don't want to constrain the dynamic evolution of the artifact so, informally, the only intra-artifact constraints we will consider are those needed for consistency. ■

6.2 Dynamic Constraints Formalism

We turn to the dynamic constraints formalism, used both to specify intra and inter dynamic constraints of artifact systems (including artifact lifecycles) to specify dynamic properties of processes running over relational artifact systems. Several choices are possible: here we focus on a variant of μ-calculus [Eme96], which is one of the most powerful temporal logics, which subsumes both linear time logics, such as LTL and PSL, and branching time logics such as CTL and CTL* [CGP99]. In particular, we introduce a variant of μ-calculus, called $\mu\mathcal{L}$ that conforms with the basic assumption of our formalism, namely the use of range-restricted FOL queries, i.e., open formulas over a fixed set of constants, to talk about the information contained in the instances.

Formally, $\mu\mathcal{L}$ formulas over $\mathcal{A}$ have the form

$$\Phi ::= Q \mid \neg\Phi \mid \Phi_1 \wedge \Phi_2 \mid \Phi_1 \vee \Phi_2 \mid \exists x \in \mathit{const}\mathcal{I}_0.\Phi \mid \forall x \in \mathit{const}\mathcal{I}_0.\Phi \mid [\Phi] \mid \langle\Phi\rangle \mid \mu Z.\Phi \mid \nu Z.\Phi \mid Z,$$

where Q is a possibly open FOL formula over the relations in the artifacts of $\mathcal{A}$, and Z is a second order predicate variable. The symbols μ and ν can be considered as quantifiers, and we make use of the notions of scope, bound and free occurrences of variables, closed formulas, etc. Definitions of these notions are as in FOL, treating μ and ν as quantifiers. In fact, we are interested only in closed formulas as specifications of temporal properties to verify.

For formulas of the form $\mu Z.\Phi$ and $\nu Z.\Phi$, we require the *syntactic monotonicity* of Φ wrt Z: every occurrence of the variable Z in Φ must be within the scope of an even number of negation signs. In $\mu\mathcal{L}$, given the requirement of syntactic monotonicity, the least fixpoint $\mu Z.\Phi$ and the greatest fixpoint $\nu Z.\Phi$ always exist.

To define the meaning of a $\mu\mathcal{L}$ formula over an artifact system, we resort to transition systems. Let $\mathfrak{A}$ be a transition system generated by a given progression mechanism over the artifact system $\mathcal{A}$. We denote by $\Sigma_{\mathfrak{A}}$ the states of $\mathfrak{A}$, and by $\mathit{const}\mathfrak{A}$ all terms (which are in general infinite) occurring in any state of $\mathfrak{A}$. Notice that trivially $\mathit{const}\mathcal{I}_0 \subseteq \mathit{const}\mathfrak{A}$.

Let $\mathcal{V}$ be a predicate and individual variable valuation on $\mathfrak{A}$, i.e., a mapping from the predicate variables Z to subsets of the states $\Sigma_{\mathfrak{A}}$, and from individual variables to constants in constA. Then, we assign meaning to $\mu\mathcal{L}$ formulas by associating to $\mathfrak{A}$ and $\mathcal{V}$ an *extension function* $(\cdot)^{\mathfrak{A}}_{\mathcal{V}}$, which maps $\mu\mathcal{L}$ formulas to subsets of $\Sigma_{\mathfrak{A}}$. The extension function $(\cdot)^{\mathfrak{A}}_{\mathcal{V}}$ is defined inductively as shown in Figure 3, where $Q\mathcal{V}$ (resp., $Z\mathcal{V}$) denotes the application of variable valuation $\mathcal{V}$ to query Q (resp., variables Z), and $\mathit{ans}(Q\mathcal{V}, \mathcal{I})$ denotes the result of evaluating the (boolean) query $Q\mathcal{V}$ over the instance $\mathcal{I}$. Moreover, $\mathcal{I} \xrightarrow{\mathcal{I}}{}'$ holds iff the progression mechanism allows to progress from $\mathcal{I}$ to $\mathcal{I}'$.

$$\begin{aligned}
(\neg\Phi)^{\mathfrak{A}}_{\mathcal{V}} &= \Sigma_{\mathfrak{A}} - (\Phi)^{\mathfrak{A}}_{\mathcal{V}}\\
(\Phi_1 \wedge \Phi_2)^{\mathfrak{A}}_{\mathcal{V}} &= (\Phi_1)^{\mathfrak{A}}_{\mathcal{V}} \cap (\Phi_2)^{\mathfrak{A}}_{\mathcal{V}}\\
(\Phi_1 \vee \Phi_2)^{\mathfrak{A}}_{\mathcal{V}} &= (\Phi_1)^{\mathfrak{A}}_{\mathcal{V}} \cup (\Phi_2)^{\mathfrak{A}}_{\mathcal{V}}\\
(\exists x \in \mathit{const}\mathcal{I}_0.\Phi)^{\mathfrak{A}}_{\mathcal{V}} &= \bigcup\{(\Phi)^{\mathfrak{A}}_{\mathcal{V}[x/c]} \mid c \in \mathit{const}\mathcal{I}_0\}\\
(\forall x \in \mathit{const}\mathcal{I}_0.\Phi)^{\mathfrak{A}}_{\mathcal{V}} &= \bigcap\{(\Phi)^{\mathfrak{A}}_{\mathcal{V}[x/c]} \mid c \in \mathit{const}\mathcal{I}_0\}
\end{aligned}$$

$$\begin{aligned}
(Z)^{\mathfrak{A}}_{\mathcal{V}} &= Z\mathcal{V} \subseteq \Sigma_{\mathfrak{A}}\\
(Q)^{\mathfrak{A}}_{\mathcal{V}} &= \{\mathcal{I} \in \Sigma_{\mathfrak{A}} \mid \mathit{ans}(Q\mathcal{V}, \mathcal{I})\}\\
(\langle\Phi\rangle)^{\mathfrak{A}}_{\mathcal{V}} &= \{\mathcal{I} \in \Sigma_{\mathfrak{A}} \mid \exists \mathcal{I}'.\, \mathcal{I} \xrightarrow{\mathcal{I}}{}' \text{ and } \mathcal{I}' \in (\Phi)^{\mathfrak{A}}_{\mathcal{V}}\}\\
([\Phi])^{\mathfrak{A}}_{\mathcal{V}} &= \{\mathcal{I} \in \Sigma_{\mathfrak{A}} \mid \forall \mathcal{I}'.\, \mathcal{I} \xrightarrow{\mathcal{I}}{}' \text{ implies } I' \in (\Phi)^{\mathfrak{A}}_{\mathcal{V}}\}\\
(\mu Z.\Phi)^{\mathfrak{A}}_{\mathcal{V}} &= \bigcap\{\mathcal{E} \subseteq \Sigma_{\mathfrak{A}} \mid (\Phi)^{\mathfrak{A}}_{\mathcal{V}[Z/\mathcal{E}]} \subseteq \mathcal{E}\}\\
(\nu Z.\Phi)^{\mathfrak{A}}_{\mathcal{V}} &= \bigcup\{\mathcal{E} \subseteq \Sigma_{\mathfrak{A}} \mid \mathcal{E} \subseteq (\Phi)^{\mathfrak{A}}_{\mathcal{V}[Z/\mathcal{E}]}\}
\end{aligned}$$

Figure 3 – Semantics of $\mu\mathcal{L}$ formulas

Intuitively, the extension function $(\cdot)^{\mathfrak{A}}_{\mathcal{V}}$ assigns to the various $\mu\mathcal{L}$ constructs the following meanings: The boolean connectives have the expected meaning, while (individual) quantification involving transitions from some state to the next is restricted to constants of $constI_0$. The extension of $\langle\Phi\rangle$ consists of the states $\mathcal{I}$ such that for *some* state $\mathcal{I}'$ with $\mathcal{I} \xrightarrow{\mathcal{I}}{}'$, we have that Φ holds in $\mathcal{I}'$, while the extension of $[\Phi]$ consists of the states $\mathcal{I}$ such that for *all* states $\mathcal{I}'$ with $\mathcal{I} \xrightarrow{\mathcal{I}}{}'$, we have that Φ holds in $\mathcal{I}'$. The extension of $\mu Z.\Phi$ is the *smallest subset* $\mathcal{E}_\mu$ of $\Sigma_{\mathfrak{A}}$ such that, assigning to Z the extension $\mathcal{E}_\mu$, the resulting extension of Φ is contained in $\mathcal{E}_\mu$. That is, the extension of $\mu Z.\Phi$ is the *least fixpoint* of the operator $(\Phi)^{\mathfrak{A}}_{\mathcal{V}[Z/\mathcal{E}]}$ (here $\mathcal{V}[Z/\mathcal{E}]$ denotes the predicate valuation obtained from $\mathcal{V}$ by forcing the valuation of Z to be $\mathcal{E}$). Similarly, the extension of $\nu X.\Phi$ is the *greatest subset* $\mathcal{E}_\nu$ of $\Sigma_{\mathfrak{A}}$ such that, assigning to X the extension $\mathcal{E}_\nu$, the resulting extension of Φ contains $\mathcal{E}_\nu$. That is, the extension of $\nu X.\Phi$ is the *greatest fixpoint* of the operator $(\Phi)^{\mathfrak{A}}_{\mathcal{V}[X/\mathcal{E}]}$. When Φ is a closed formula, $(\Phi)^{\mathfrak{A}}_{\mathcal{V}}$ does not depend on $\mathcal{V}$, and we denote it by $\Phi^{\mathfrak{A}}$.

We say that a closed $\mu\mathcal{L}$ formula Φ holds for $\mathfrak{A}$, denoted as $\mathfrak{A} \models \Phi$, iff $\mathcal{I}_0 \in \Phi^{\mathfrak{A}}$. We call *model checking* verifying whether $\mathfrak{A} \models \Phi$ holds.

Example 2 (Continues from Example 1)*:* Now that we have defined our constraints formalism, we are in the position to express the constraints informally discussed in Example 1.

For ReqOrders, we first define formulas corresponding to the phases of the diagram in Figure 2a:

$$\begin{aligned}
\psi_A &= \neg\exists x, y, z.\mathsf{ROItem}(x, y, z)\\
\psi_B &= \forall x, y, z.(\mathsf{ROItem}(x, y, z) \rightarrow z = \mathsf{requested}) \wedge \exists x, y.\mathsf{ROItem}(x, y, \mathsf{requested})\\
\psi_C &= \ \forall x, y, z.(\mathsf{ROItem}(x, y, z) \rightarrow (z = \mathsf{requested} \vee z = \mathsf{purchased})) \wedge\\
&\quad \exists x, y.\mathsf{ROItem}(x, y, \mathsf{requested}) \wedge \exists x, y.\mathsf{ROItem}(x, y, \mathsf{purchased})\\
\psi_D &= \forall x, y, z.(\mathsf{ROItem}(x, y, z) \rightarrow z = \mathsf{purchased}) \wedge \exists x, y.\mathsf{ROItem}(x, y, \mathsf{purchased})\\
\psi_E &= \forall x, y, z.(\mathsf{ROItem}(x, y, z) \rightarrow (z = \mathsf{purchased} \vee z = \mathsf{shipped})) \wedge\\
&\quad \exists x, y.\mathsf{ROItem}(x, y, \mathsf{purchased}) \wedge \exists x, y.\mathsf{ROItem}(x, y, \mathsf{shipped})\\
\psi_F &= \forall x, y, z.(\mathsf{ROItem}(x, y, z) \rightarrow z = \mathsf{shipped}) \wedge \exists x, y.\mathsf{ROItem}(x, y, \mathsf{shipped}).
\end{aligned}$$

Then, the dynamic constraints of ReqOrders are captured by the formula

$$\Phi_{RO} = \psi_A \wedge \nu Z.(\textstyle\bigwedge_{i=1,\ldots,11} \Phi_i \wedge \Box Z).$$

It requires that in the initial state of $\mathfrak{A}$ there are not any items included in any pending order, i.e., the relation ROItem is empty, and that all formulas Φ_i listed below hold in every state. Each of Φ_1 to Φ_6 corresponds to a single transition as in Figure 2a, expressing the constraint that the artifact remains in its current phase *until* it reaches the following one, also requiring that such a phase is eventually reached in a finite number of steps, and that no other phase is reached until then:

$$\begin{array}{rclrcl}
\Phi_1 &=& \psi_A \rightarrow \mu Z.(\psi_B \vee (\psi_A \wedge \Box Z)) & \Phi_4 &=& \psi_D \rightarrow \mu Z.(\psi_E \vee (\psi_D \wedge \Box Z)) \\
\Phi_2 &=& \psi_B \rightarrow \mu Z.(\psi_C \vee (\psi_B \wedge \Box Z)) & \Phi_5 &=& \psi_E \rightarrow \mu Z.(\psi_F \vee (\psi_E \wedge \Box Z)) \\
\Phi_3 &=& \psi_C \rightarrow \mu Z.(\psi_D \vee (\psi_C \wedge \Box Z)) & \Phi_6 &=& \psi_F \rightarrow \mu Z.(\psi_A \vee (\psi_F \wedge \Box Z)).
\end{array}$$

The remaining formulas express static constraints, specifically inclusion dependencies and range restrictions:

$$\begin{array}{rcl}
\Phi_7 &=& \forall x, y, z.(\mathsf{ROItem}(x, y, z) \rightarrow \exists u, v.\mathsf{RO}(x, u, v)) \\
\Phi_8 &=& \forall x, y, z.(\mathsf{ROItem}(x, y, z) \rightarrow \mathsf{Status}(z)) \\
\Phi_9 &=& \forall x.(\mathsf{Status}(x) \rightarrow (x = \mathsf{requested} \vee x = \mathsf{purchased} \vee x = \mathsf{shipped})) \\
\Phi_{10} &=& \forall x, y, z.(\mathsf{ROItem}(x, y, z) \rightarrow \exists w \mathsf{LineItem}(y, w)) \\
\Phi_{11} &=& \forall x, y, z.(\mathsf{RO}(x, y, z) \rightarrow (\mathsf{Requester}(y) \wedge \mathsf{Buyer}(z)).
\end{array}$$

For ProcOrders, we just need to express some static specifications over instances. Hence, Φ_{PO} is the conjunction of the following formulas, expressing inclusion dependency constraints:

$$\begin{array}{l}
\nu Z.(\forall x, y, z.(\mathsf{PO}(x, y, z) \rightarrow \mathsf{Supplier}(z)) \wedge \Box Z) \\
\nu Z.(\forall x, y, z.(\mathsf{POItem}(x, y, z) \rightarrow \exists u.\mathsf{PO}(x, y, u)) \wedge \Box Z) \\
\nu Z.(\forall x, y, z.(\mathsf{POItem}(x, y, z) \rightarrow \exists u.\mathsf{LineItem}(u, z)) \wedge \Box Z).
\end{array}$$

Finally, the set of inter-artifact dynamic constraints Φ_{inter} is the conjunction of the following formulas:

$$\begin{array}{l}
\nu Z.(\forall x, y.(\mathsf{ReqOrders.LineItem}(x, y) \leftrightarrow \mathsf{ProcOrders.LineItem}(x, y)) \wedge \Box Z) \\
\nu Z.(\forall x, y, z.(\mathsf{POItem}(x, y, z) \rightarrow \mathsf{ROItem}(y, z, \mathsf{purchased})) \wedge \Box Z) \\
\nu Z.(\forall x, y, z.(\mathsf{PO}(x, y, z) \rightarrow \exists w, k.\mathsf{RO}(y, w, k)) \wedge \Box Z).
\end{array}$$

The first formula requires that the LineItem relations in both artifacts have the same set of tuples, the second one that every item belonging to a procurement order is also included in some requisition order, and the third one that every procurement order corresponds to a requisition order. ■

6.3 Processes over Artifact Systems

We now concentrate on progression mechanisms for relational artifact systems. In particular, we specify such a mechanism in terms of one or more *processes* that use actions as atomic steps. *Actions* represent atomic tasks or services that act over the artifacts and make them evolve.

6.3.1 Actions

We give a formal specification of actions in terms of preconditions and postconditions [HR99], inspired by the notion of mapping in the literature on data exchange [Kol05]. However, we generalize such a notion in order to include negation, arbitrary quantification in preconditions, and the generation of new terms, through the use of *Skolem functions* in postconditions. Notice that, while it is conceivable that most of the actions will act on one artifact only, we do not make such a restriction. Indeed our actions are generally inter-artifact, which lets us easily account for synchronisation between artifacts.

An *action* ρ for $\mathcal{A}$ has the form

$$\rho(p_1, \ldots, p_m) : \{e_1, \ldots, e_m\} \qquad \text{where:}$$

- $\rho(p_1, \ldots, p_m)$ is the *signature* of the action, constituted by a name ρ and a sequence $p_1, \ldots, p_m$ of *input parameters* that need to be substituted by constants for the execution of the action, and

- $\{e_1, \ldots, e_m\}$ is a set of effects, called the *effects' specification.*

We denote by σ a (ground) substitution for the input parameters with terms not involving variables. Given such a substitution σ, we denote by $\rho\sigma$ the action with actual parameters. All effects in the effects' specification are assumed to take place simultaneously. Specifically, an *effect* e_i has the form

$$q_i^+ \wedge Q_i^- \rightsquigarrow I_i' \qquad \text{where:}$$

- $q_i^+ \wedge Q_i^-$ is a query whose terms are variables $\vec{x}$, action parameters, and constants from $const\mathcal{I}_0$. The query q_i^+ is a UCQ, and the query Q_i^-, is an arbitrary FOL formula whose free variables are included in those of q_i^+. Intuitively, q_i^+ selects the tuples to instantiate the effect, and Q_i^- filters away some of them.

- I_i' is a set of facts for the artifacts in $\mathcal{A}$, which includes as terms: terms in $const\mathcal{I}_0$, input parameters, free variables of q_i^+, and in addition terms formed by applying an arbitrary Skolem function to one of the previous kinds of terms. Such Skolem terms are used as witnesses of values chosen by the external user/environment when executing the action. Notice that different effects can share a same Skolem function.

Given an instance $\mathcal{I}$ of $\mathcal{A}$, an effect e_i as above, and a substitution σ for the parameters of e_i, the effect e_i extracts from $\mathcal{I}$ the set $ans((q_i^+ \wedge Q_i^-)\sigma, \mathcal{I})$ of tuples of terms, and for each such tuple θ asserts the set $I_i'\sigma\theta$ of facts obtained from $I_i'\sigma$ by applying the substitution θ for the free variables of q_i^+. In particular, in the resulting set of facts we may have terms of the form $f(\vec{t})\sigma\theta$ where $\vec{t}$ is a set of terms that may be either free variables in $\vec{x}$, parameters, or terms in $const\mathcal{I}_0$. We denote by $e_i\sigma(\mathcal{I})$ the overall set of facts, i.e., $e_i\sigma(\mathcal{I}) = \bigcup_{\theta \in ans((q_i^+ \wedge Q_i^-)\sigma, \mathcal{I})} I_i'\sigma\theta$. The overall *effect* of the action ρ with parameter substitution σ over $\mathcal{I}$ is a new instance $\mathcal{I}' = do(\rho\sigma, \mathcal{I}) = \bigcup_{1 \leq i \leq m} e_i\sigma(\mathcal{I})$ for A.

Some observations are in order: *(i)* In the formalization above actions are *deterministic*, in the sense that, given an instance $\mathcal{I}$ of $\mathcal{A}$ and a substitution σ for the parameters of an action ρ, there is a *single* instance $\mathcal{I}'$ that is obtained as the result of executing ρ in $\mathcal{I}$. *(ii)* The effects of an action are naturally a form of update of the previous state, and not of belief revision [KM91]. That is, we never learn new facts on the state in which an action is executed, but only on the state resulting from the action execution. *(iii)* We do not make any persistence (or frame) assumption in our formalization [Rei01]. In principle at every move we substitute the whole old state, i.e., instance, $\mathcal{I}$, with a new one, $\mathcal{I}'$. On the other hand, it should be clear that we can easily write effect specifications that *copy* big chunks of the old state into the new one. For example, $R_i(\vec{x}) \rightsquigarrow R_i(\vec{x})$ copies the entire set of assertions involving the relation R_i.

6.3.2 Processes

Essentially processes are (possibly nondeterministic) programs that use artifacts in $\mathcal{A}$ to store their (intermediate and final) computation results, and use actions in $\boldsymbol{\rho}$ as atomic instructions. We assume that at every time the current instance $\mathcal{I}$ can be arbitrarily queried through the query answering services, while it can be updated only through the actions in $\boldsymbol{\rho}$. Notice that, while we require the execution of actions to be sequential, we do not impose any such constraints on processes, which in principle can be formed by several concurrent branches, including fork, join, and so on. Concurrency is to be interpreted by interleaving, as often done in formal verification [CGP99, Eme96]. There can be many ways to provide the control flow specification for processes for $\mathcal{A}$. Here we adopt a very simple rule-based mechanism. Notice, however, that our results can be immediately generalized to any process formalism whose processes control flow is finite-state. Notice also that the transition system associated to a process over an artifact might not be finite-state, since its state is formed by both the *control flow* state of the process and the *data* in the artifact system, which are in general unbounded.

Formally, a process Π over a relational artifact system $\mathcal{A}$ is a pair $\langle \boldsymbol{\rho}, \boldsymbol{\pi} \rangle$, where $\boldsymbol{\rho}$ is a finite set of actions and $\boldsymbol{\pi}$ is a finite set of condition-action rules.

A *condition-action rule* π in $\boldsymbol{\pi}$ is an expression of the form

$$Q \mapsto \rho,$$

where ρ is an action in $\boldsymbol{\rho}$ and Q is a FOL formula over artifacts' relations whose free variables are exactly the parameters of ρ, and whose other terms can be either quantified variables or terms in $const\mathcal{I}_0$. Such a rule has the following semantics: for each tuple σ for which condition Q holds, the action ρ with actual parameters σ *can* be executed. If ρ has no parameters then Q will be a boolean formula. Observe that processes don't force the execution of actions but constrain them: the user of the process will be able to choose any of the actions that the rules forming the process allow.

The *execution* of a process Π over a relational artifact system $\mathcal{A}$ is defined as follows: we start from $\mathcal{I}_0$, and for each rule $Q \mapsto \rho$ in Π, we evaluate Q, and for each tuple σ returned, we execute $\rho\sigma$, obtaining a new instance $\mathcal{I}' = do(\rho\sigma, \mathcal{I}_0)$, and so on. In this way we build a *transition system* $\Upsilon(\Pi, \mathcal{A})$ whose states represent possible system instances, and where each transition represents the execution of an instantiated action that is allowed according to the process. A transition $\mathcal{I} \xrightarrow{[} \Upsilon(\Pi, \mathcal{A})]\mathcal{I}'$ holds iff there exists a rule $Q \mapsto \rho$ in Π such that there exists a $\sigma \in ans(Q, \mathcal{I})$ and $\mathcal{I}' = do(\rho\sigma, \mathcal{I})$. That is, there exist a rule in Π that can fire on $\mathcal{I}$ and produce an instantiated action $\rho\sigma$, which applied on $\mathcal{I}$, results in $\mathcal{I}'$.

The transition system $\Upsilon(\Pi, \mathcal{A})$ captures the behavior of the process Π over the whole system $\mathcal{A}$. We are interested in formally verifying properties of processes over artifact-based systems, in particular we are interested in *conformance* and *verification*, defined as follows:

Conformance. Given a process Π and an artifact system $\mathcal{A}$, the process is said to be acceptable if it fulfills all intra-artifact and inter-artifact dynamic constraints. In this case, we say that Π *conforms to* $\mathcal{A}$. In order to formally check *conformance*, we can resort to model checking and verify that:

$$\Upsilon(\Pi, \mathcal{A}) \;\models\; \Phi_{inter} \wedge \bigwedge_{i=1,\ldots,n} \Phi_i.$$

Verification. Apart from intra-artifacts and inter-artifact dynamic constraints, we are interested in other dynamic properties of the process over the artifact system. We say that a process Π over an artifact system $\mathcal{A}$ *verifies* a dynamic property Φ expressed in $\mu\mathcal{L}$ if

$$\Upsilon(\Pi, \mathcal{A}) \models \Phi.$$

It becomes evident that model checking of the transition system $\Upsilon(\Pi, \mathcal{A})$ generated by a process over an artifact system is the critical form of reasoning needed in our framework. We are going to study such a reasoning task next.

Example 3 (Continues from Example 2)*:* We consider a process $\Pi = \langle \boldsymbol{\rho}, \boldsymbol{\pi} \rangle$ constituted by the following actions $\boldsymbol{\rho}$ and conditions-action rules $\boldsymbol{\pi}$. When specifying an action, we will use $[\ldots]$ to delimit each of the two parts q_i^+ and Q_i^- of the formula $q_i^+ \wedge Q_i^-$ in the left-hand side of an effect specification. Note that in such a formula the part corresponding to Q_i^- might be missing.

Actions. The set $\vec{\rho}$ of actions is the following. Action $\mathsf{RequestItem}(r,i,b)$ is used by the requester r to request a new line item i to buyer b. Such an action results in adding i to the requisition order of r. Notice that the *RoCode* denoting the requisition order is computed as a function of r and b only: performing this action multiple times for the same requester and buyer will result into adding line items to the same requisition order.

$$\begin{array}{ll}\mathsf{RequestItem}(r,i,b): & \{\ [\exists w.(\mathsf{Requester}(r) \wedge \mathsf{LineItem}(i,w) \wedge \mathsf{Buyer}(b))] \rightsquigarrow \\ & \quad \{\mathsf{RO}(f(r,b),r,b),\ \mathsf{ROItem}(f(r,b),i,\mathsf{requested})\}, \\ & \ \ \mathit{CopyAll}\ \}\end{array}$$

Action $\mathsf{Purchase}(r,i,b,s)$ is used by buyer b for purchasing an item i belonging to requisition order r from supplier s, thus creating (or updating) procurement orders (i.e., the relation $\mathsf{ProcOrders.PO}$) and updating the status of the corresponding items kept by the relation $\mathsf{ReqOrders.ROItem}$. Again, notice that *PoCode* is not a function of the item i passed as parameter. By writing $\mathit{CopyAll} \setminus \mathsf{ROItem}$ we denote the copy of all relations except ROItem.

$$\begin{array}{ll}\mathsf{Purchase}(r,i,b,s): & \{\ [\exists w.\mathsf{RO}(r,w,b) \wedge \mathsf{ROItem}(r,i,\mathsf{requested}) \wedge \mathsf{Supplier}(s)] \rightsquigarrow \\ & \quad \{\mathsf{PO}(g(r,b,s),r,s),\ \mathsf{POItem}(g(r,b,s),r,i), \\ & \quad\ \ \mathsf{ROItem}(r,i,\mathsf{purchased})\}, \\ & \ \ [\mathsf{ROItem}(x,y,z)] \wedge [\neg\mathsf{ROItem}(r,i,\mathsf{requested})] \rightsquigarrow \\ & \quad \{\mathsf{ROItem}(x,y,z)\}, \\ & \ \ \mathit{CopyAll} \setminus \mathsf{ROItem}\ \}\end{array}$$

The following actions are used to ship all items included in a given procurement order p, and to deliver items belonging to a requisition order r to the corresponding requester, respectively. Notice that the first avoids copying all facts concerning p whereas the latter does the same with all facts related to r.

$$\begin{array}{ll}\mathsf{Ship}(p): & \{\ [\mathsf{POItem}(p,x,y) \wedge \exists z.\mathsf{ROItem}(x,y,z)] \rightsquigarrow \{\mathsf{ROItem}(x,y,\mathsf{shipped})\}, \\ & \ \ [\mathsf{POItem}(x,y,z)] \wedge [\neg\mathsf{POItem}(p,y,z)] \rightsquigarrow \{\mathsf{POItem}(x,y,z)\}, \\ & \ \ [\mathsf{PO}(x,y,z)] \wedge [\neg\mathsf{PO}(p,y,z)] \rightsquigarrow \{\mathsf{PO}(x,y,z)\}, \\ & \ \ \mathit{CopyAll} \setminus (\mathsf{POItem} \text{ and } \mathsf{PO})\ \} \\ \mathsf{Deliver}(r): & \{\ [\mathsf{ROItem}(x,y,z)] \wedge [\neg\mathsf{ROItem}(r,y,z)] \rightsquigarrow \{\mathsf{ROItem}(x,y,z)\}, \\ & \ \ [\mathsf{RO}(x,y,z)] \wedge [\neg\mathsf{RO}(r,y,z)] \rightsquigarrow \{\mathsf{RO}(x,y,z)\}, \\ & \ \ \mathit{CopyAll} \setminus (\mathsf{ROItem} \text{ and } \mathsf{RO})\ \}\end{array}$$

Condition-action rules. In each condition-action rule of our process, we instantiate the parameters passed to the action, while simply checking that they are meaningful, i.e., that they are in the current instance. Hence:

$$\begin{array}{ll}\boldsymbol{\pi} = \{ & \exists x.(\mathsf{Requester}(r) \wedge \mathsf{LineItem}(i,x) \wedge \mathsf{Buyer}(b)) \mapsto \mathsf{RequestItem}(r,i,b), \\ & \exists x.(\mathsf{RO}(r,x,b) \wedge \mathsf{ROItem}(r,i,\mathsf{requested}) \wedge \mathsf{Supplier}(s)) \mapsto \mathsf{Purchase}(r,i,b,s), \\ & \exists x,y.\mathsf{PO}(p,x,y) \mapsto \mathsf{Ship}(p), \\ & \exists x,y.\mathsf{RO}(r,x,y) \mapsto \mathsf{Deliver}(r)\ \}\end{array}$$

We close our example by observing that the process we have specified conforms to the lifecycle in Example 2. ■

6.4 Undecidability of Conformance and Verification

Next, we consider conformance and verification over relational artifact systems. We show that they are both undecidable in general, even in simple cases. The undecidability result does not come as a surprise, since the transition system of a process over an artifact system can easily be infinite-state. Moreover, our framework is so general that it does not force the infinite state space regularity usually needed to apply known results on model checking on infinite state systems. However, we show that the undecidability holds even in a very simple case.

We consider a relational artifact system of the form $\mathcal{A}_u = \langle \{A\}, \top \rangle$ with $A = \langle \vec{R}, I_0, \top \rangle$. That is $\mathcal{A}_u$ is formed by a single artifact A with no intra-artifact or inter-artifact dynamic constraints. In addition, we consider processes with only one action ρ_u and only one condition-action rule $\top \mapsto \rho_u$ that has a $\top$ condition and hence allows the execution of the action ρ_u at every moment. The action ρ_u is without parameters, its effects have the form

$$q_i^+ \rightsquigarrow I_i',$$

where q_i^+ is a CQ (hence without any form of negation and of universal quantification), and it includes *CopyAll* effects. We call these kinds of relational artifact systems and processes *simple*. Next lemma shows that it is undecidable to verify in such cases the $\mu\mathcal{L}$ formula $\mu Z.(q \vee \langle Z \rangle)$, expressing that there exists a sequence of action executions that leads to an instance where a boolean CQ q holds.

Lemma 11. *Verifying whether the $\mu\mathcal{L}$ formula $\mu Z.(q \vee \langle Z \rangle)$ holds for a simple process over a simple artifact is undecidable.*

sketch. We observe that we can use the set of effects of ρ_u to encode a set of tuple-generating dependencies (TGDs) [AHV95b]. Hence we can reduce to the above verification problem the problem of answering boolean CQs in a relational database under a set of TGDs, which is undecidable [BV81]. (In fact, special care is needed because of the use of Skolem terms instead of labeled nulls.) □

Theorem 9. *Conformance checking and verification are both undecidable for processes over relational artifacts systems.*

sketch. Lemma 11 gives us undecidability of verification, already for simple relational artifact systems and processes. To get undecidability of conformance it is sufficient to consider the simple process Π_u over relational artifact systems of the form $\mathcal{A}_{cu} = \langle \{A_c\}, \top \rangle$, with $A_c = \langle \vec{R}, I_0, \mu Z.(q \vee \langle Z \rangle) \rangle$. Note that $\mathcal{A}_{cu}$ is a variant of simple artifact systems $\mathcal{A}_u$ in which the artifact has as intra-artifact dynamic constraint exactly $\mu Z.(q \vee \langle Z \rangle)$. The claim follows again from Lemma 11, considering that, by definition, checking conformance of the simple process Π_u wrt $\mathcal{A}_{cu}$ is equivalent to checking whether $\Upsilon(\Pi_u, \mathcal{A}_u) \models \mu Z.(q \vee \langle Z \rangle)$. □

6.5 Decidability of Weakly Acyclic Processes

Next we tackle decidability, and, inspired by the recent literature on data exchange [Kol05], we isolate a notable case of processes over relational artifact systems for which both conformance and verification are decidable. Our results rely on the possibility of building a special process that we call "positive approximate". For such a process there exists a tight correspondence between the application of an action and a step in the chase of a set of TGDs [AHV95b, Kol05]

Given a process $\Pi = \langle \boldsymbol{\rho}, \boldsymbol{\pi} \rangle$, the *positive approximate* of Π is the process $\Pi^+ = \langle \boldsymbol{\rho}^+, \boldsymbol{\pi}^+ \rangle$ obtained from Π as follows. For each action ρ in $\boldsymbol{\rho}$, there is an action ρ^+ in $\boldsymbol{\rho}^+$, obtained from ρ by

- removing all input parameters from the signature, and

- substituting each effect $q_i^+ \wedge Q_i^- \rightsquigarrow I_i'$ with the one that uses only the *positive* part of the head of the effect specification, i.e., with $q_i^+ \rightsquigarrow I_i'$.

Note that the variables in q_i^+ that used to be parameters in ρ, become free variables in ρ^+. Then, for each condition-action rule $Q \mapsto \rho$ in $\boldsymbol{\pi}$, there is a rule $\top \mapsto \rho^+$ in $\boldsymbol{\pi}^+$. Hence, Π^+ allows for executing every action at every step.

Now, relying again on the parallelism between chase in data exchange and action execution in artifact systems, we take advantage of the notion of weak acyclicity in data exchange [Kol05] to devise an interesting class of processes which are guaranteed to generate a finite-state transition system, when run over a relational artifact system. This in turn guarantees decidability of conformance and verification.

Let Π be a process over an artifact system $\mathcal{A}$, and $\Pi^+ = \langle \boldsymbol{\rho}^+, \boldsymbol{\pi}^+ \rangle$ its positive approximate. We call *dependency graph* of Π^+ the following (edge labeled) directed graph:

Nodes: for every artifact $A = \langle \boldsymbol{R}, I_0, \Phi \rangle$ of $\mathcal{A}$, every relation symbol $R_i \in \boldsymbol{R}$, and every attribute att or R_i, there is a node (R_i, att) representing a position;

Edges: for every action ρ^+ of $\boldsymbol{\rho}^+$, every effect $q_i^+(\boldsymbol{t}) \rightsquigarrow I_i'(\boldsymbol{t}', f_1(\boldsymbol{t}_1), \ldots, f_n(\boldsymbol{t}_n))$ of ρ^+ (where for convenience we have made explicit the terms occurring in q_i^+ and I_i', and where consequently $\boldsymbol{t}', \boldsymbol{t}_1, \ldots, \boldsymbol{t}_n \subseteq \boldsymbol{t}$ are either constants or variables), every variable $x \in \boldsymbol{t}$, and every occurrence of x in q_i^+ in position p, there are the following edges:

- for every occurrence of x in I_i' in position p', there is an edge $p \rightarrow p'$;
- for every Skolem term $f_k(t_k)$ such that $x \in \boldsymbol{t}_k$ occurs in I_i' in position p'', there is a *special edge* (i.e., one labeled by $*$) $p \xrightarrow{*} p''$.

We say that Π is *weakly acyclic* if the dependency graph of Π^+ has no cycle going through a special edge.

Intuitively, ordinary edges keeps track of the fact that a value may propagate from position p to position p' in a possible trace. Moreover, special edges keeps track of the fact that a value in position p can be taken as parameter of a Skolem function, thus contributing to the creation of a (not necessarily new) value in any position p''. If a cycle goes through a special edge, then a new value appearing in a certain position may determine the creation of another one, in the same position, later during the execution of actions. Since this may happen again and again, no bound can be put on the number of newly generated Skolem terms, and thus on the number of new values appearing in the instance. Note that the definition allows for cycles as long as they do not include special edges.

Lemma 12. *Let* Π *be a* weakly acyclic *process over a relational artifact system* $\mathcal{A}$ *with initial instance* $\mathcal{I}_0$*, and let* Π^+ *be the positive approximate of* Π*. Then there exists a polinomial in the size of* $\mathcal{I}_0$ *that bounds the size of every instance generated by* Π^+*.*

sketch. The proof follows the line of that in [Kol05] on chase termination for weakly acyclic TGDs. The difference here is that we use Skolem terms and don't have the inflationary behavior of TGDs in applying action effects. However, the key notion of rank used in [Kol05] can still be used to bound the number of terms generated through the Skolem functions. □

Notice that as a direct result of this lemma, the transition system generated by the positive approximate over $\mathcal{A}$ has a number of states that is finite, and in fact at most exponential in the size of the initial instance $\mathcal{I}_0$ of $\mathcal{A}$. Now we show that a similar result holds for the original process Π. The key to this is the following observation that easily follows from the definition of ρ^+ for an action ρ.

Lemma 13. *For every action ρ over $\mathcal{A}$, instances $\mathcal{I}_1$, $\mathcal{I}_2$ of $\mathcal{A}$, and ground substitution σ for the parameters of ρ, if $\mathcal{I}_1 \subseteq \mathcal{I}_2$ then $do(\rho\sigma, \mathcal{I}_1) \subseteq do(\rho^+, \mathcal{I}_2)$.*

We can extend the result above to any sequence of actions, by induction on the length of the sequence. Hence, we get that the instance obtained from the initial instance by executing a sequence of actions of the original process Π is contained in the instance obtained by executing the same sequence of actions of Π^+. From this observation, considering the bound in Lemma 12, we get the desired result for the original process.

Lemma 14. *Let* Π *be a* weakly acyclic *process over a relational artifact system $\mathcal{A}$ with initial instance $\mathcal{I}_0$. Then there exists a polinomial in the size of $\mathcal{I}_0$ that bounds the size of every instance generated by* Π.

From this, we obtain our main result.

Theorem 10. *Conformance and verification of $\mu\mathcal{L}$ formulas are decidable for weakly acyclic processes over relational artifact systems.*

sketch. From Lemma 14, it follows that the transition system generated by a weakly acyclic process over a relational artifact system $\mathcal{A}$ has a number of states that is at most exponential in the size of the initial instance $\mathcal{I}_0$ of $\mathcal{A}$. The claim then follows from known results on verification of μ-calculus formulas over finite transition systems (see e.g., [Eme96]). □

From the exponential bound on the number of states of the generated transition system mentioned in the proof above, we get not only decidability of verification and conformance, but also an ExpTime upper bound for its computational complexity (assuming a bound on the nesting of fixpoints).

6.6 Discussion

In this Section we have looked at foundations of artifact-centric systems, and we have shown that weakly acyclic processes over relational artifacts are very interesting both from a formal point of view, since reasoning on them is decidable, and from a practical point of view, since weak-acyclicity appears to be a quite acceptable restriction.

Further research can take several directions. First, one can easily focus on different temporal logics for specifying dynamic constraints, such as LTL or CTL. Observe that the results presented here would apply, being μ-calculus more expressive than both LTL and CTL, but certainly they can be refined. Second, we may introduce special equality generating constraints to allow to equate different terms, e.g., a Skolem term and a constant. We are particularly interested in how to extend our decidability result to this case. Also we have assumed that no artifacts are added or destroyed during the execution of a process. We are very interested in overcoming this assumption. Notice that to do so we would need to introduce Skolem terms to denote artifacts, and then extend the notion of weakly acyclic process to block the infinite accumulation of new artifacts. Finally, we are interested in moving from a relational setting to a semantic one, based on ontologies for data access [CDGL+11], believing that similar results apply.

7 Synthesizing Agent Protocols From LTL Specifications against Multiple Partially-Observable Environments

A key component of any intelligent agent is its ability of reasoning about the actions it performs in order to achieve a certain goal. If we consider a single-agent interacting with an environment, a natural question to ask is the extent to which an agent can derive a plan to achieve a given goal. Under the assumption of full observability of the environment, the methodology of LTL synthesis enables the automatic generation, e.g., through a model checker, of a set of rules for the agent to achieve a goal expressed as an LTL specification. This is a well-known decidable setting but one that is 2EXPTIME-complete [PR89b, KV00] due to the required determinisation of non-deterministic Büchi automata. Solutions that are not complete but computationally attractive have been put forward [HRS05]. Alternative approaches focus on subsets of LTL, e.g., GR(1) formulas as in [PPS06a].

Work in AI on planning has of course also addressed this issue albeit from a rather different perspective. The emphasis here is most often on sound but incomplete heuristics that are capable of generating effective plans on average. Differently from main-stream synthesis approaches, a well-explored assumption here is that of partial information, i.e., the setting where the environment is not fully observable by the agent. Progress here has been facilitated by the relatively recent advances in the efficiency of computation of Bayesian expected utilities and Markov decision processes. While these approaches are attractive, differently from work in LTL-synthesis, they essentially focus on goal reachability [BG09].

An automata-based approach to planning for full-fledged LTL goals covering partial information was put forward in [DV99] where an approach based on non-emptiness of Büchi-automata on infinite words was presented. An assumption made in [DV99] is that the agent is interacting with a single deterministic environment which is only partially observable. It is however natural to relax this assumption, and study the case where an agent has the capability of interacting with multiple, partially-observable, environments sharing a common interface. A first contribution in this direction was made in [HD11] which introduced generalized planning under multiple environments for reachability goals and showed that its complexity is EXPSPACE-complete. The main technical results of this section is to give a sound and complete procedure for solving the generalized planning problem for LTL goals, within the same complexity bound.

A further departure from [HD11] is that we here ground the work on interpreted systems [FHMV95, PR85], a popular agent-based semantics. This enables us to anchor the framework to the notion of *agent protocol*, i.e., a function returning the set of possible actions in a given local state, thereby permitting implementations on top of existing model checkers [GvdM04, LQR09a]. As as already remarked in the literature [vdM96], interpreted systems are particularly amenable to incorporating observations and uncertainty in the environment. The model we pursue here differentiates between visibility of the environment states and observations made by the agent (given what is visible). Similar models have been used in the past, e.g, the VSK model discussed in [WL01]. Differently from the VSK model, here we are not concerned in epistemic specifications nor do we wish to reason at specification level about what is visible, or observable. The primary aim, instead, is to give sound and complete procedures for solving the generalized planning problem for LTL goals.

With a formal notion of agent, and agent protocol at hand, it becomes natural to distinguish several ways to address generalized planning for LTL goals, namely:

- *State-based solutions*, where the agent has the ability of choosing the "right" action toward the achievement of the LTL goal, among those that its protocol allows, exploiting the current observation, but avoiding memorizing previous observations. So the resulting plan is essentially a *state-based strategy*, which can be encoded in the agent protocol itself. While this solution is particularly simple, is also often too restrictive.

- *History-based solutions*, where the agent has the ability of remembering all observations made in the past, and use them to decide the "right" action toward the achivement of the LTL goal, again among those allowed by its protocol. In this case we get a *perfect-recall strategy*. These are the most general solutions, though in principle such solutions could be infinite and hence unfeasible. One of the key result of this section however is that if a perfect-recall strategy exists, then there exist one which can be represented with a finite number of states.

- *Bounded solutions*, where the agent decides the "right" course of actions, by taking into account only a fragment of the observation history. In this case the resulting strategies can be encoded as a new agent protocol, still compatible with the original one, though allowing the internal state space of the original agent to grow so as to incorporate the fragment of history to memorize. Since we show that if a perfect-recall strategy exists, there exist one that is finite state (and hence makes use only of a fragment of the history), we essentially show that wlog we can restrict our attention to bounded solution (for a suitable bound) and incorporate such solutions in the agent protocol.

The rest of the section is organized as follows. First we give the formal framework for our investigations, and we look at state-based solutions. Then, we move to history based solutions and prove the key technical results of this section. We give a sound, complete, and computationally optimal (wrt worst case complexity) procedure for synthesizing perfect-recall strategies from LTL specification. Moreover, we observe that the resulting strategy can be represented with finite states. Then, we show how to encode such strategies (which are in fact bounded) into agent protocol. Finally, we briefly look at an interesting variant of the presented setting where the same results hold.

The material in this work has been published in [FdGL12].

7.1 Framework

Much of the literature on knowledge representation for multi-agent systems employs modal temporal-epistemic languages defined on semantic structures called *interpreted systems* [FHMV95, PR85]. These are refined versions of Kripke semantics in which the notions of computational states and actions are given prominence. Specifically, all the information an agent has at its disposal (variables, facts of a knowledge base, observations from the environment, etc.) is captured in the *local state* relating to the agent in question. *Global states* are tuples of local states, each representing an agent in the system as well as the environment. The environment is used to represent information such as messages in transit and other components of the system that are not amenable to an intention-based representation.

An interesting case is that of a single agent interacting with an environment. This is not only interesting in single-agent systems, or whenever we wish to study the single-agent interaction, but it is also a useful abstraction in loosely-coupled systems where all of the agents' interactions take place with the environment. Also note that the modular reasoning approach in verification focuses on the single agent case in which the environment encodes the rest of the system and its potential influence to the component under analysis. In this and other cases it is useful to model the system as being composed by a single agent only, but interacting with multiple environments, each possibly modelled by a different finite-state machine.

With this background we here pursue a design paradigm enabling the refinement of a generic agent program following a planning synthesis procedure given through an LTL specification, such that the overall design and encoding is not affected nor invalidated. We follow the interpreted system model and formalise an agent as being composed of (i) a set of local states, (ii) a set of actions, (iii) a *protocol* function specifying which actions are available in each local state, (iv) an observation function encoding what perceptions are being made by the agent, and (v) a local evolution function. An environment is modelled similarly to an agent; the global transition function is defined on the local transition functions of its components, i.e., the agent's and the environment's.

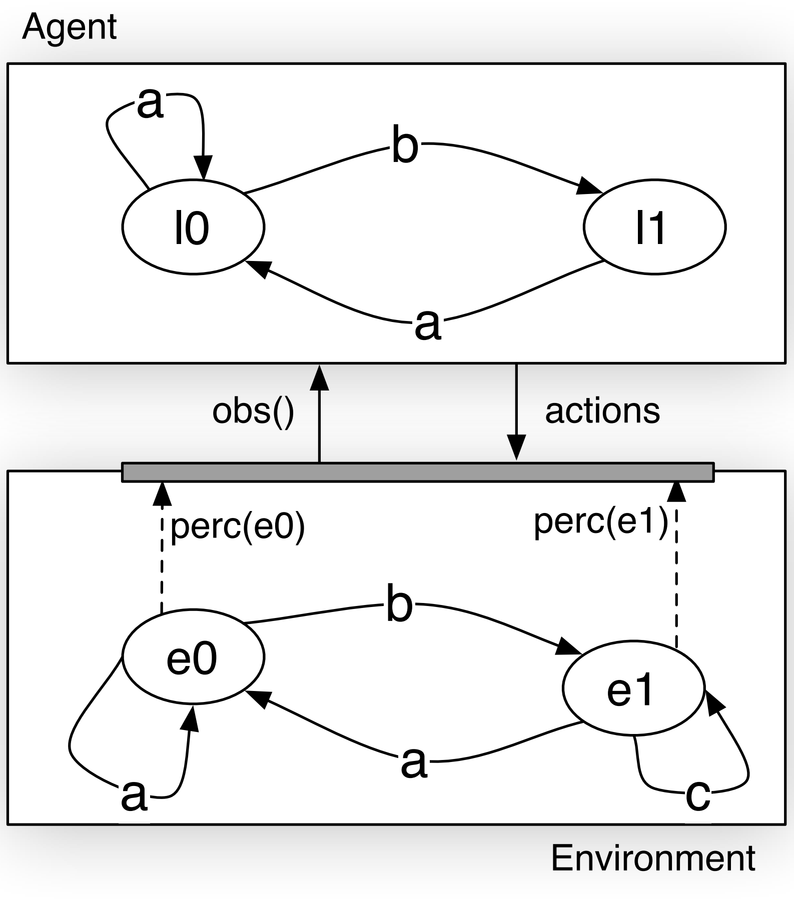


Figure 4 – Interaction between *Ag* and *Env*

We refer to Figure 4 for a pictorial description of our setting in which the agent is executing actions on the environment, which, in turn, respond to these actions by changing its state. The observations of the agent depend on what perceived of the states of the environment. Notice that environments are only partially observable through what perceivable of their states. So two different environments may give rise to the same observations in the agent.

Environment.

An *environment* is a tuple $Env = \langle E, Act_e, Perc, perc, \delta, e_0 \rangle$ such that:

- $E = \{e_0, \ldots\}$ is a finite set of local states of the environment;
- Act_e is the alphabet of the environment's actions;
- $Perc$ is the alphabet of perceptions;
- $perc : E \rightarrow Perc$ is the perceptions (output) function;
- $\delta : E \times Act_e \rightarrow E$ is a (partial) transition function;
- $e_o \in E$ is the initial state.

Notice that, as in classical planning, such an environment is deterministic.

A *trace* is a sequence $trace = e_0\alpha_1 e_1 \alpha_2 \ldots \alpha_n e_n$ of environment's states such that $e_{i+1} = \delta(e_i, \alpha_{i+1})$ for each $0 \leq i < n$. The corresponding *perceivable* trace is the trace obtained by applying the perception function: $perc(trace) = perc(e_0), \ldots, perc(e_n)$.

Similarly, an agent is represented as a finite machine, whose state space is obtained by considering the agent's internal states, called *configurations*, together with all additional information the agent can access, i.e., the observations it makes. We take the resulting state space as the agent's *local states*.

Agent.

An *agent* is a tuple $Ag = \langle Conf, Act_a, Obs, obs, L, poss, \delta_a, c_0 \rangle$ where:

- $Conf = \{c_0, \ldots\}$ is a finite set of the agent's configurations (internal states);
- Act_a is the finite set of agent's actions;
- Obs is the alphabet of observations the agent can make;
- $obs : Perc \rightarrow Obs$ is the observation function;
- $L = Conf \times Obs$ is the set of agent's local states;
- $poss : L \rightarrow \wp(Act_a)$ is a protocol function;
- $\delta_a : L \times Act_a \rightarrow Conf$ is the (partial) transition function;
- $c_0 \in Conf$ is the initial configuration. A local state $l = \langle c, o \rangle$ is called *initial* iff $c = c_0$.

Agents defined as above are deterministic: given a local state l and an action α, there exists a unique next configuration $c' = \delta_a(l, \alpha)$. Nonetheless, observations introduce non-determinism when computing the new local state resulting from action execution: executing action α in a given local state $l = \langle c, o \rangle$ results into a new local state $l' = \langle c', o' \rangle$ such that $c' = \delta_a(l, \alpha)$ and o' is the new observation, which can not be foreseen in advance.

Consider the agent and the environment depicted in Figure 4. Assume that the agent is in its initial configuration c_0 and that the current environment's state is e_0. Assume also that $obs(perc(e_0)) = o$, i.e., the agent receives observation o. Hence, the current local state is $l_0 = \langle c_0, o \rangle$. If the agent performs action b (with $b \in poss(l_0)$), the agent moves from configuration c_0 to configuration $c_1 = \delta_a(\langle c_0, o \rangle, b)$. At the same time, the environment changed its state from e_0 to e_1, so that the new local state is $l_1 = \langle c_1, o' \rangle$, where $o' = obs(perc(e_1))$.

Notice that the protocol function is not defined with respect to the transition function, i.e., according to transitions available to the agent. In fact, we can imagine an agent having its own behaviours, in terms of transitions defined over configurations, that can be constrained according to some protocol, which can in principle be modified or substituted. Hence, we say that a protocol is *compatible* with an agent iff it is compatible with its transition function, i.e., $\alpha \in poss(\langle c, o\rangle) \to \exists c' \in Conf \mid \delta_a(\langle c, o\rangle, \alpha) = c'$. Moreover, we say that a protocol *poss* is an *action-deterministic protocol* iff it always returns a singleton set, i.e., it allows only a single action to be executed for a given local state. Finally, an agent is *nonblocking* iff it is equipped with a compatible protocol function *poss* and for each sequence of local states $l_0\alpha_1 l_1\alpha_2 \dots \alpha_n l_n$ such that $l_i = \langle c_i, o_i\rangle$ and $\alpha_{i+1} \in poss(\langle c_i, o_i\rangle)$ for each $0 \le i < n$, we have $poss(l_n) \neq \emptyset$. So, an agent is nonblocking iff it has a compatible protocol function which always provides a non-empty set of choices for each local state that is reachable according to the transition function and the protocol itself.

Finally, given a perceivable trace of an environment *Env*, the *observation history* of an agent *Ag* is defined as the sequence obtained by applying the observation function: $obs(perc(trace)) = obs(perc(e_0)), \dots, obs(perc(e_n))$. Given one such history $h \in Obs^*$, we denote with $last(h)$ its latest observation: $last(h) = obs(perc(e_n))$.

7.2 LTL Specifications against Multiple Environments

In this section, we consider an *agent Ag* and a *finite set $\mathcal{E}$ of environments*, all sharing common actions and the same perception alphabet. Such environments are indistinguishable by the agent, in the sense that the agent is not able to identify which environment it is actually interacting with, unless through observations. The problem we address is thus to synthesize a (or customize the) agent protocol so as to fulfill a given LTL specification (or goal) in all environments. A planning problem for an LTL goal is a triple $\mathcal{P} = \langle Ag, \mathcal{E}, \mathcal{G}\rangle$, where Ag is an agent, $\mathcal{E}$ an environment set, and $\mathcal{G}$ an LTL goal specification. This setting is that of *generalized planning* [HD11], extended to deal with *long-running goals*, expressed as arbitrary LTL formulae. This is also related to *planning for* LTL *goals* under partial observability [DV99].

Formally, we call *environment set* a finite set of environments $\mathcal{E} = \{Env_1, \dots, Env_k\}$, with $Env_i = \langle E_i, Act_{ei}, Perc, perc_i, \delta_i, e_{0i}\rangle$. Environments share the same alphabet *Perc* of the agent *Ag*. Moreover the *Ag*'s actions must be executable in the various environments: $Act_a \subseteq \bigcap_{i=1,\dots,k} Act_{ei}$. This is because we are considering an agent acting in environments with similar "interface".

As customary in verification and synthesis [BK08], we represent an LTL goal with a Büchi automaton[10] $\mathcal{G} = \langle Perc, G, g_0, \gamma, G^{acc}\rangle$, describing the desired behaviour of the system in terms of perceivable traces, where:

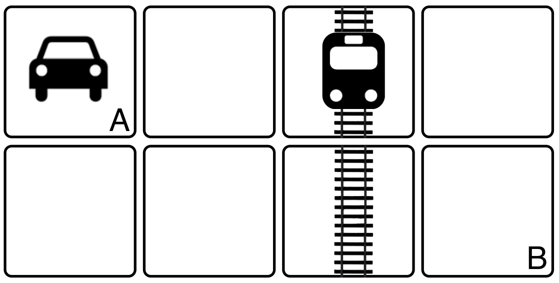
- *Perc* is the finite alphabet of perceptions, taken as input;
- G is a finite set of states;
- g_0 is the initial state;
- $\gamma : G \times Perc \to 2^G$ is the transition function;

[10] In fact, while any LTL formula can be translated into a Büchi automaton, the converse is not true. Our results hold for any goal specified as a Büchi automaton, though for ease of exposition we give them as LTL.

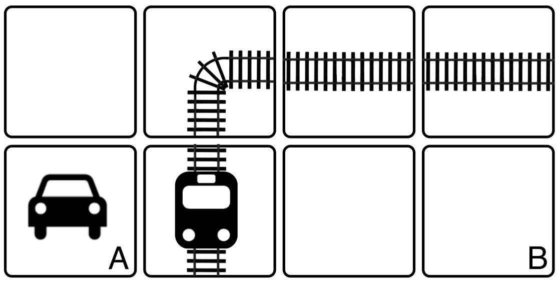
- $G^{acc} \subseteq G$ is the set of accepting states.

A run of $\mathcal{G}$ on an input word $w = p_0, p_1, \ldots \in Perc^{\omega}$ is an infinite sequence of states $\rho = g_0, g_1, \ldots$ such that $g_i \in \gamma(g_{i-1}, p_i)$, for $i > 0$. Given a run ρ, let $inf(\rho) \subseteq G$ be the set of states occurring infinitely often, i.e., $inf(\rho) = \{g \in G \mid \forall i\ \exists j > i \text{ s.t. } g_j = g\}$. An infinite word $w \in Perc^{\omega}$ is accepted by $\mathcal{G}$ iff there exists a run ρ on w such that $inf(\rho) \cap G^{acc} \neq \emptyset$, i.e., at least one accepting state is visited infinitely often during the run. Notice that, since the alphabet $Perc$ is shared among environments, the same goal applies to all of them. As we will see later, we can characterize a variant of this problem by considering the environment's internal states as $\mathcal{G}$'s alphabet.

Example 4*:* Consider a simple environment Env_1 constituted by a grid of 2x4 cells, each denoted by e_{ij}, a train, and a car. A railroad crosses the grid, passing on cells e_{13}, e_{23}. Initially, the car is in e_{11} and the train in e_{13}. The car is controlled by the agent, whereas the train is a moving obstacle moving from e_{13} to e_{23} to e_{13} again and so on. The set of actions is $Act_{e1} = \{goN, goS, goE, goW, wait\}$. The train and the car cannot leave the grid, so actions are allowed only when feasible. The state space is then $E_1 = \{e_{11}, \ldots, e_{24}\} \times \{e_{13}, e_{23}\}$, representing the positions of the car and the train. We consider a set of perceptions $Perc = \{\texttt{posA}, \texttt{posB}, \texttt{danger}, \texttt{dead}, \texttt{nil}\}$, and a function $perc_1$ defined as follows: $perc_1(\langle e_{11}, e_t\rangle) = \texttt{posA}$, $perc_1(\langle e_{24}, e_t\rangle) = \texttt{posB}$, $perc_1(\langle e_{13}, e_{23}\rangle) = perc_1(\langle e_{23}, e_{13}\rangle) = \texttt{danger}$ and $perc_1(\langle e_{13}, e_{13}\rangle) = perc_1(\langle e_{23}, e_{23}\rangle) = \texttt{dead}$. $perc_1(\langle e_c, e_t\rangle) = \texttt{nil}$ for any other state.



(a) Env_1



(b) Env_2

Figure 5 – Environments Env_1 and Env_2

We consider a second environment Env_2 similar to Env_1 as depicted in Figure 5b. We skip details about its encoding, as it is analogous to Env_1.

Then, we consider a third environment Env_3 which is a variant of the Wumpus world [RN10], though sharing the interface (in particular they all share perceptions and actions) with the other two. Following the same convention used before, the hero is initially in cell e_{11}, the Wumpus in e_{31}, gold is in e_{34} and the pit in e_{23}. The set of actions is $Act_{e3} = \{goN, goS, goE, goW, wait\}$. Recall that the set of perceptions $Perc$ is instead shared with Env_1 and Env_2. The state space is $E_3 = \{e_{11}, \ldots, e_{34}\}$, and the function $perc_3$ is defined as follows: $perc_3(e_{11}) = \texttt{posA}$, $perc_3(e_{34}) = \texttt{posB}$, $perc_3(e_{23}) = perc_3(e_{31}) = \texttt{dead}$, $perc_3(e) = \texttt{danger}$ for $e \in \{e_{13}, e_{21}, e_{22}, e_{24}, e_{32}, e_{33}\}$, whereas $perc_3(e) = \texttt{nil}$ for any other state. This example allows us to make some observation about our framework. Consider first the perceptions $Perc$. They are intended to represent signals coming from the environment, which is modeled as a "black box". If we could distinguish between perceptions (instead of having just a `danger` perception), we would be able to identify the current environment as Env_3, and solve such a problem separately. Instead, in our setting the perceptions are not informative enough to discriminate environments (or the agent is not able to observe them); so all environments need to be considered together. Indeed, Env_3 is similar to Env_1 and Env_2 at the interface level, and it is attractive to try to synthesize a single strategy to solve them all. In some sense, crashing in Env_1 or Env_2 corresponds to falling into the pit or being eaten by the Wumpus; the same holds for danger states with the difference that perceiving `danger` in Env_1 or Env_2 can not be used to prevent an accident.

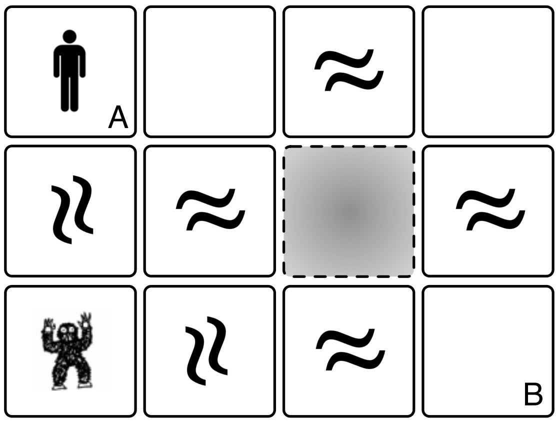


Figure 6 – Environment Env_3

Notice that in our framework we design the environments without taking into account the agent Ag that will interact with them. Likewise, the same holds when designing Ag. Indeed, agent Ag is encoded as an automaton with a single configuration c_0, and all actions being loops. In particular, let $Act_a = Act_{ei}$, $i = 1, 2, 3$. Notice also that, by suitably defining the observation function *obs*, we can model the agent's sensing capabilities, i.e., its ability to observe perceptions coming from the environment. Suppose that Ag can sense perceptions $\texttt{posA}, \texttt{danger}, \texttt{dead}$, but it is unable to sense `posB`, i.e., its sensors can not detect such signal. To account for this, it is enough to consider the observation function *obs* as a "filter" (i.e. $Obs \subseteq Perc$), such that $Obs = \{\texttt{posA}, \texttt{danger}, \texttt{dead}, \texttt{nil}\}$ and $obs(\texttt{posA}) = \texttt{posA}$, $obs(\texttt{danger}) = \texttt{danger}$, $obs(\texttt{dead}) = \texttt{dead}$, and $obs(\texttt{nil}) = obs(\texttt{posB}) = \texttt{nil}$. Since Ag is filtering away perception `posB`, the existence of a strategy does not imply that agent Ag is actually able to recognize that it has achieved its goal. Notice that this does not affect the solution, nor its executability, in any way. The goal is achieved irrespective of what the agent can observe.

Moreover, Ag has a "safe" protocol function *poss* that allows all moving actions to be executed in any possible local state, but prohibits it to perform *wait* if the agent is receiving the observation `danger`: $poss(\langle c_0, o\rangle) = Act_a$ if $o \neq \texttt{danger}$, $Act_a \setminus \{wait\}$ otherwise.

Finally, let $\mathcal{G}$ be the automaton corresponding to the LTL formula $\phi_{\mathcal{G}} = (\Box\Diamond\texttt{posA}) \wedge (\Box\Diamond\texttt{posB}) \wedge \Box\neg\texttt{dead}$ over the perception alphabet, constraining perceivable traces such that the controlled objects (the car / the hero) visit positions A and B infinitely often. ■

7.3 State-Based Solutions

To solve the synthesis problem in the context above, the first solution that we analyze is based on customizing the agent to achieve the LTL goal, by *restricting the agent protocol* while keeping the same set of local states. We do this by considering so called state-based strategies (or plans) to achieve the goal. We call a strategy for an agent Ag *state-based* if it can be expressed as a (partial) function

$$\sigma_p : (Conf \times Obs) \to Act_a$$

For it to be acceptable, a strategy also needs to be *allowed* by the protocol: it can only select available actions, i.e., for each local state $l = \langle c, o \rangle$ we have to have $\sigma_p(l) \in poss(l)$.

State-based strategies do not exploit an agent's memory, which, in principle, could be used to reduce its uncertainty about the environment by remembering observations from the past. Exploiting this memory requires having the ability of extending its configuration space, which at the moment we do not allow (see later). In return, these state-based strategies can be synthesized by just taking into account all allowed choices the agent has in each local state (e.g., by exhaustive search, possibly guided by heuristics). The advantage is that to meet its goal, the agent Ag does not need any modification to its configurations, local states and transition function, since only the protocol is affected. In fact, we can see a strategy σ_p as a restriction of an agent's protocol yielding an action-deterministic protocol $\overline{poss}$ derived as follows:

$$\overline{poss}(\langle c, o \rangle) = \begin{cases} \{\alpha\}, & \text{iff } \sigma_p(c, o) = \alpha \\ \emptyset, & \text{if } \sigma_p(c, o) \text{ is undefined} \end{cases}$$

Notice that $\overline{poss}$ is then a total function. Notice also that agent $\overline{Ag}$ obtained by substituting the protocol function maintains a protocol compatible with the agent transition function. Indeed, the new allowed behaviours are a subset of those permitted by original agent protocol.

Example 5*:* Consider again Example 4. No state-based solution exists for this problem, since selecting the same action every time the agent is in a given local state does not solve the problem. Indeed, just by considering the local states we can not get any information about the train's position, and we would be also bound to move the car (the hero) in the same direction every time we get the same observation (agent Ag has only one configuration c_0). Nonetheless, observe that if we could keep track of past observations when selecting actions, then a solution can be found.

7.4 History-Based Solutions

We turn to strategies that can take into account past observations. Specifically, we focus on strategies (plans) that may depend on the entire unbounded observation history. These are often called perfect-recall strategies.

A *(perfect-recall) strategy* for $\mathcal{P}$ is a (partial) function

$$\sigma : Obs^* \to Act_a$$

that, given a sequence of observations (the *observation history*), returns an action. A strategy σ is *allowed* by Ag's protocol iff, given any observation history $h \in Obs^*$, $\sigma(h) = \alpha$ implies $\alpha \in poss(\langle c, last(h) \rangle)$, where c is the current configuration of the agent. Notice that, given an observation history, the current configuration can be always reconstructed by applying the transition function of Ag, starting from initial configuration c_0. Hence, a strategy σ is a solution of the problem $\mathcal{P} = \langle Ag, \mathcal{E}, \mathcal{G} \rangle$ iff it is allowed by Ag and it generates, for each environment Env_i, an infinite trace $\tau = e_{0i}\alpha_1 e_{1i}\alpha_2 \ldots$ such that the corresponding perceivable trace $perc(\tau)$ satisfies the LTL goal, i.e., it is accepted by the corresponding Büchi automaton.

The technique we propose is based on previous automata theoretic approaches. In particular, we extend the technique for automata on infinite strings presented in [DV99] for partial observability, to the case of a finite set of environments, along the lines of [HD11]. The crucial point is that we need both the ability of simultaneously dealing with LTL goals and with multiple environments. We build a generalized Büchi automaton that returns sequences of *action vectors* with one component for each environment in the environment set $\mathcal{E}$. Assuming $|\mathcal{E}| = k$, we arbitrarily order the k environments, and consider sequences of action vectors of the form $\vec{a} = \langle a_1, \ldots, a_k \rangle$, where each component specifies one operation for each environment. Such sequences of action vectors correspond to a strategy σ, which, however, must be *executable*: for any pair $i, j \in \{1, \ldots, k\}$ and observation history $h \in Obs^*$ such that both σ_i and σ_j are defined, then $\sigma_i(h) = \sigma_j(h)$. In other words, if we received the same observation history, the function select the same action. In order to achieve this, we keep an *equivalence relation* $\equiv \subseteq \{1, \ldots, k\} \times \{1, \ldots, k\}$ in the states of our automaton. Observe that this equivalence relation has correspondences with the epistemic relations considered in epistemic approaches [JvdH04, LR06, PS11].

We are now ready to give the details of the automata construction. Given a set of k environments $\mathcal{E}$ with $Env_i = \langle E_i, Act_{ei}, Perc, perc_i, \delta_i, e_{0i} \rangle$, an agent $Ag = \langle Conf, Act_a, Obs, obs, L, poss, \delta_a, c_0 \rangle$ and goal $\mathcal{G} = \langle Perc, G, g_0, \gamma, G^{acc} \rangle$, we build the generalized Büchi automaton $\mathcal{A}_\mathcal{P} = \langle Act_a^k, W, w_0, \rho, W^{acc} \rangle$ as follows:

- $Act_a^k = (Act_a)^k$ is the set of k-vectors of actions;
- $W = E^k \times Conf^k \times G^k \times \wp(\equiv)$;[11]
- $w_0 = \langle e_{10}, \ldots, e_{k0}, c_0, \ldots, c_0, g_0, \ldots, g_0, \equiv_0 \rangle$ where

  $$i \equiv_0 j \text{ iff } obs(perc_i(e_{i0})) = obs(perc_j(e_{j0}));$$

- $\langle \vec{e}', \vec{c}', \vec{g}', \equiv' \rangle \in \rho(\langle \vec{e}, \vec{c}, \vec{g}, \equiv \rangle, \vec{\alpha})$ iff
  - if $i \equiv j$ then $\alpha_i = \alpha_j$;
  - $e'_i = \delta_i(e_i, \alpha_i)$;
  - $c'_i = \delta_a(l_i, \alpha_i) \wedge \alpha_i \in poss(l_i)$
    where $l_i = \langle c_i, obs(perc_i(e_i)) \rangle$;
  - $g'_i = \gamma(g_i, perc_i(e_i))$;
  - $i \equiv' j$ iff $i \equiv j \wedge obs(perc_i(e'_i)) = obs(perc_j(e'_j))$.
- $W^{acc} = \{$
  $E^k \times Conf^k \times G^{acc} \times G \times \ldots \times G \times \equiv$, ... ,
  $E^k \times Conf^k \times G \times \ldots \times G \times G^{acc} \times \equiv \}$

Each automaton state $w \in W$ holds information about the internal state of each environment, the corresponding current goal state, the current configuration of the agent for each environment, and the equivalence relation. Notice that, even with fixed agent and goal, we need to duplicate their corresponding components in each state of $\mathcal{A}_\mathcal{P}$ in order to consider all possibilities for the k environments. In the initial state w_0, the environments, the agent and the goal automaton are in their respective initial state. The initial equivalence relation $\equiv_0$ is computed according to the observation provided by environments. The transition relation ρ is built by suitably composing the transition function of each state component, namely δ_a for agent, δ_i for the environments, and γ for the goal. Notice that we do not build transitions in which an action α is assigned to the agent when either it is in a configuration from which a transition with action α is not defined, or α is not allowed by the protocol *poss* for the current local state. The equivalence relation

[11]We denote by $\wp(\equiv)$ the set of all possible equivalence relations $\equiv \subseteq \{1, \ldots, k\} \times \{1, \ldots, k\}$.

is updated at each step by considering the observations taken from each environment. Finally, each member of the accepting set W^{acc} contains a goal accepting state, in some environment.

Once this automaton is constructed, we check it for non-emptiness. If it is not empty, i.e., there exists a infinite sequence of action vectors accepted by the automaton, then from such an infinite sequence it is possible to build a strategy realizing the LTL goal. The non-emptiness check is done by resolving polynomially transforming the generalized Büchi automaton into standard Büchi one and solving *fair reachability* over the graph of the automaton, which (as standard reachability) can be solved in NLOGSPACE [Var96]. The non-emptiness algorithm itself can also be used to return a strategy, if it exists.

The following result guarantees that not only the technique is sound (the perfect-recall strategies do realize the LTL specification), but it is also complete (if a perfect-recall strategy exists, it will be constructed by the technique).

this technique is correct, in the sense that if a perfect-recall strategy exists then it will return one.

Theorem 11 (Soundness and Completeness)**.** *A strategy σ that is a solution for problem $\mathcal{P} = \langle Ag, \mathcal{E}, \mathcal{G}\rangle$ exists iff $\mathcal{L}(\mathcal{A}_\mathcal{P}) \neq \emptyset$.*

Proof. ($\Leftarrow$) The proof is based on the fact that $\mathcal{L}(\mathcal{A}_\mathcal{P}) = \emptyset$ implies that it holds the following persistence property: for all runs in $\mathcal{A}_\mathcal{P}$ of the form $r_\omega = w_0\vec{\alpha_1}w_1\vec{\alpha_2}w_2 \ldots \in W^\omega$ there exists an index $i \geq 0$ such that $w_j \notin W^{acc}$ for any $j \geq i$. Conversely, if $\mathcal{L}(\mathcal{A}_\mathcal{P}) \neq \emptyset$, there exists an infinite run $r_\omega = w_0\vec{\alpha_1}w_1\vec{\alpha_2}w_2 \ldots$ on $\mathcal{A}_\mathcal{P}$ visiting at least one state for each accepting set in W^{acc} infinitely often (as required by its acceptance condition), thus satisfying the goal in each environment. First, we notice that such an infinite run r_ω is the form $r_\omega = r'(r'')^\omega$ where both r' and r'' are finite sequences. Hence such a run can be represented with a finite *lazo* shape representation: $r = w_0\vec{a}_1w_1 \ldots \vec{a}_nw_n\vec{a}_{n+1}w_m$ with $m < n$ [Var96]. Hence we can synthesize the corresponding partial function σ by unpacking r (see later). Essentially, given one such r and any observable history $h = o_0, \ldots, o_\ell$ and denoting with α_{ji} the i-th component of $\vec{\alpha}_j$, σ is inductively defined as follows:

- if $\ell = 0$ then $\sigma(o_0) = \alpha_{1i}$ iff $o_0 = obs(perc_i(e_{0i}))$ in w_0.
- if $\sigma(o_0, \ldots, o_{\ell-1}) = \alpha$ then $\sigma(h) = \alpha_{ji}$ iff $o_\ell = obs(perc_i(e_{ji}))$ in $w_j = \langle \vec{e}_j, \vec{c}_j, \vec{g}_j, \equiv_j\rangle$ and α is such that $\alpha = \alpha_{\ell z}$ with $i \equiv_j z$ for some z, where $j = \ell$ if $\ell \leq m$, otherwise $j = m + \ell$ mod(n-m). If instead $o_\ell \neq obs(perc_i(e_{ji}))$ for any e_{ji} then $\sigma(h)$ is undefined.

Indeed, σ is a prefix-closed function of the whole history: we need to look at the choice made at previous step to keep track of it. In fact, we will see later how unpacking r will result into a sort of tree-structure representation. Moreover, it is trivial to notice that any strategy σ synthesized by emptiness checking $\mathcal{A}_\mathcal{P}$ is allowed by agent Ag. In fact, transition relation ρ is defined according to the agent's protocol function *poss*.

($\Rightarrow$) Assume that a strategy σ for $\mathcal{P}$ does exist. We prove that, given such σ, there exists in $\mathcal{A}_\mathcal{P}$ a corresponding accepting run r_ω as before. We prove that there exists in $\mathcal{A}_\mathcal{P}$ a run $r_\omega = w_0\vec{\alpha_1}w_1\vec{\alpha_2}w_2 \ldots$, with $w_\ell = \langle \vec{e}_\ell, \vec{c}_\ell, \vec{g}_\ell, \equiv_\ell\rangle \in W$, such that:

1. $(\ell = 0)$ $\sigma(obs(perc_i(e_{0i}))) = \alpha_{1i}$ for all $0 < i \leq k$;
2. $(\ell > 0)$ if $\sigma(obs(perc_i(e_{(\ell-1)i}))) = \alpha$ for some $0 < i \leq k$, then $\alpha = \alpha_{\ell i}$ and $e_{\ell i} = \delta_i(e_{(\ell-1)i}, \alpha_{\ell i})$ and $\sigma(obs(perc_i(e_{\ell i})))$ is defined;
3. r_ω is accepting.

In other words, there exists in $\mathcal{A}_\mathcal{P}$ an accepting run that is induced by executing σ on $\mathcal{A}_\mathcal{P}$ itself. Point 1 holds since, in order to be a solution for $\mathcal{P}$, the function σ has to be defined for histories constituted by the sole observation $obs(perc_i(e_{0i}))$ of any environment initial state. According to the definition of the transition relation ρ, there exists in $\mathcal{A}_\mathcal{P}$ a transition from each e_{0i} for all available actions α such that $\delta_i(e_{0i}, \alpha)$ is defined for Env_i. In particular, the transition $\langle w, \vec{\alpha}, w'\rangle$ is not permitted in $\mathcal{A}_\mathcal{P}$ iff either some action component α_i is not allowed by agent's protocol *poss* or it is not available in the environment Env_i, $0 < i \leq k$. Since σ is a solution of $\mathcal{P}$ (and thus allowed by Ag) it cannot be one of such cases. Point 2 holds for the same reason: there exists a transition in ρ for all available actions of each environment. Point 3 is just a consequence of σ being a solution of $\mathcal{P}$. $\square$

Checking wether $\mathcal{L}(\mathcal{A}_\mathcal{P}) \neq \emptyset$ can be done NLOGSPACE in the size of the automaton. Our automaton is exponential in the number of environments in $\mathcal{E}$, but its construction can be done on-the-fly while checking for non-emptiness. This give us a PSPACE upperbound in the size of the original specification with explicit states. If we have a compact representation of those, then we get an EXPSPACE upperbound. Considering that even the simple case of generalized planning for reachability goals in [HD11] is PSPACE-complete (EXPSPACE-complete considering compact representation), we obtain a worst case complexity characterization of the problem at hand.

Theorem 12 (Complexity)**.** *Solving the problem $\mathcal{P} = \langle Ag, \mathcal{E}, \mathcal{G}\rangle$ admitting perfect-recall solutions is* PSPACE-*complete (*EXPSPACE-*complete considering compact representation).*

We conclude this section by remarking that, since the agent gets different observation histories from two environments Env_i and Env_j, then from that point on it will be always possible to distinguish these. More formally, denoting with r a run in $\mathcal{A}_P$ and with $r_\ell = \langle \vec{e}_\ell, \vec{c}_\ell, \vec{g}_\ell, \equiv_\ell\rangle$ its ℓ-th state, if $i \equiv_\ell j$, then $i \equiv_{\ell'} j$ for every state $r_{\ell' < \ell}$. Hence it follows that the equivalence relation $\equiv$ is indentical for each state belonging to the same strongly connected component of $\mathcal{A}_\mathcal{P}$. Indeed, assume by contradiction that there exists some index ℓ' violating the assumption above. This implies that $\equiv_{\ell'} \subset \equiv_{\ell'+1}$. So, there exists a tuple in $\equiv_{\ell'+1}$ that is not in $\equiv_{\ell'}$. But this is impossible since, by definition, we have that $i \equiv_{\ell'+1} j$ implies $i \equiv_{\ell'} j$.

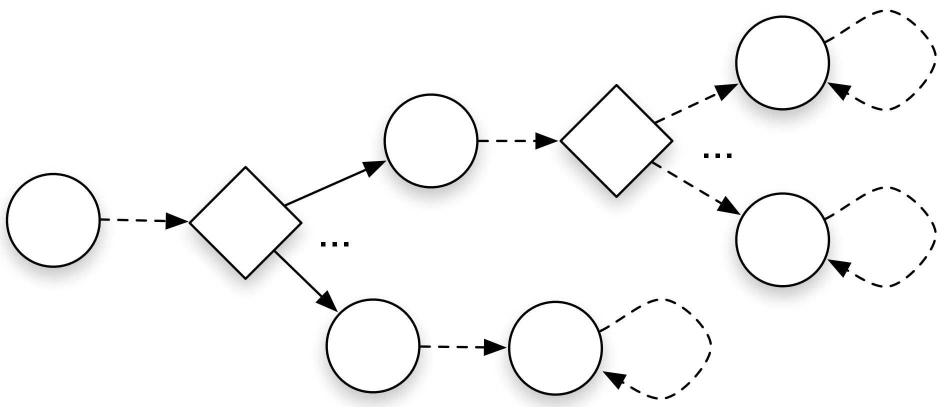


Figure 7 – Decision-tree like representation of a strategy.

Figure 7 shows a decision-tree like representation of a strategy. The diamond represents a decision point where the agent reduces its uncertainty about the environment. Each path ends in a loop thereby satisfying the automaton acceptance condition. The loop, which has no more decision point, represents also that the agent cannot reduce its uncertainty anymore and hence it has to act blindly as in conformant planning. Notice that if our environment set includes only one environment, or if we have no observations to use to reduce uncertainty, then the strategy reduces to the structure in Figure 8, which reflects directly the general *lazo* shape of runs satisfying LTL properties: a sequence reaching a certain state and a second sequence consisting in a loop over that state.

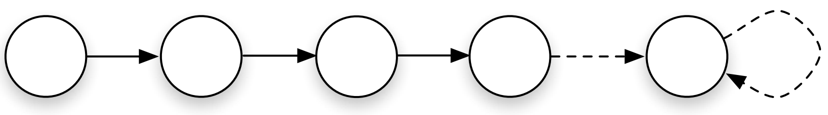
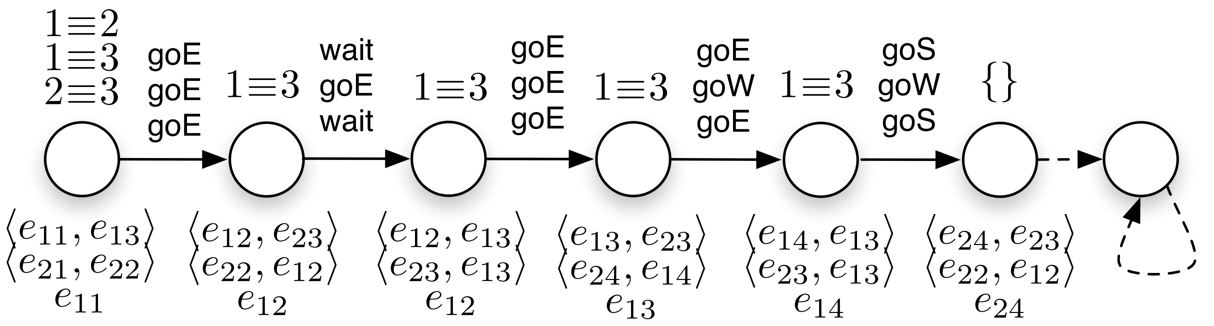


Figure 8 – A resulting strategy execution.

7.5 Representing Strategies

The technique described in the previous section provides, if a solution does exist, the run of $\mathcal{A}_\mathcal{P}$ satisfying the given LTL specification. As discussed above such a run can be represented finitely. In this section, we exploit this possibility to generate a finite representation of the run that can be used directly as the strategy σ for the agent Ag. The strategy σ can be represented as a finite-state structure with nodes labeled with agent's configuration and edges labeled with a *condition-action* rule $[o]\alpha$, where $o \in Obs$ and $\alpha \in Act_a$. The intended semantics is that a transition can be chosen to be carried on for environment Env_i only if the current observation of its internal state e_i is o, i.e. $o = obs(perc_i(e_i))$. Hence, notice that a strategy σ can be expressed by means of sequences, *case*-conditions and infinite iterations.



$1{\equiv}2$
$1{\equiv}3$
$2{\equiv}3$
goE goE goE
$1{\equiv}3$
wait goE wait
$1{\equiv}3$
goE goE goE
$1{\equiv}3$
goE goW goE
$1{\equiv}3$
goS goW goS
$\{\}$
$\langle e_{11}, e_{13}\rangle$ $\langle e_{21}, e_{22}\rangle$ e_{11}
$\langle e_{12}, e_{23}\rangle$ $\langle e_{22}, e_{12}\rangle$ e_{12}
$\langle e_{12}, e_{13}\rangle$ $\langle e_{23}, e_{13}\rangle$ e_{12}
$\langle e_{13}, e_{23}\rangle$ $\langle e_{24}, e_{14}\rangle$ e_{13}
$\langle e_{14}, e_{13}\rangle$ $\langle e_{23}, e_{13}\rangle$ e_{14}
$\langle e_{24}, e_{23}\rangle$ $\langle e_{22}, e_{12}\rangle$ e_{24}

Figure 9 – Accepting run r for Example 4.

In other words, it can be represented as a graph $G_r = \langle N, E\rangle$ where N is a set of nodes, $\lambda : N \to Conf$ its labeling, and $E \subseteq N \times \Phi \times N$ is the transition relation. G_r can be obtained by unpacking the run found as witness, exploiting the equivalence relation $\equiv$. More in details, let $r = w_0\vec{a}_1 w_1 \ldots \vec{a}_n w_n \vec{a}_{n+1} w_m$ with $m \leq n$ be a finite representation of the infinite run returned as witness. Let r_ℓ be the ℓ-th state in r, whereas we denote with $r|_\ell$ the sub-run of r starting from state r_ℓ. A projection of r over a set $X \subseteq \{1, \ldots, k\}$ is the run $r(X)$ obtained by projecting away from each w_i all vector components and indexes not in X. We mark state w_m: $loop(w_\ell) = true$ if $\ell = m$, *false* otherwise.

$$G_r = \textsc{Unpack}(r, nil);$$

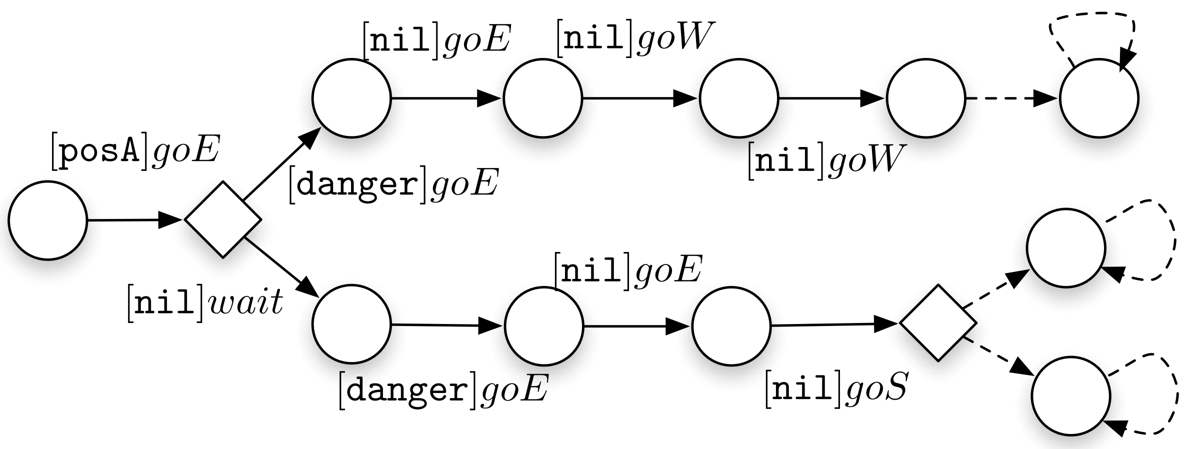


Figure 10 – Corresponding G_r

```
UNPACK($r, loopnode$):
 1: $N = E = \emptyset$;
 2: be $r_0 = \langle \vec{e}, \vec{c}, \vec{g}, \equiv \rangle$;
 3: if $loop(r_0) \wedge loopnode \neq nil$ then
 4:     return $\langle \{loopnode\}, E \rangle$;
 5: end if
 6: be $\vec{a}_1 = \langle \alpha_1, \ldots, \alpha_k \rangle$;
 7: Let $X = \{X_1, \ldots, X_b\}$ be the partition induced by $\equiv$;
 8: $node$ = new node;
 9: if $loop(r_0)$ then
10:     $loopnode = node$;
11: end if
12: for ($j = 1$; $j \leq b$; $j{+}{+}$) do
13:     $G' = $ UNPACK($r(X_j)|_1, loopnode$);
14:     choose $i$ in $X_j$;
15:     $\lambda(node) = c_i$;
16:     $E = E' \cup \langle node, [obs(perc_i(e_i))]\alpha_i, root(G') \rangle$;
17:     $N = N \cup N'$;
18: end for
19: return $\langle N \cup \{node\}, E \rangle$;
```

The algorithm above, presented in pseudocode, recursively processes run r until it completes the loop on w_m, then it returns. For each state, it computes the partition induced by relation $\equiv$ and, for each element in it, generates an edge in G_r labeled with the corresponding action α taken from the current action vector.

From G_r we can derive *finite state strategy* $\sigma_f = \langle N, succ, act, n_0 \rangle$. where:

- $succ : N \times Obs \times Act_a \rightarrow N$ such that $succ(n, o, a) = n'$ iff $\langle n, [o]\alpha, n' \rangle \in E$;
- $act : N \times Conf \times Obs \rightarrow Act_a$ such that $\alpha = act(n, c, o)$ iff $\langle n, [o]\alpha, n' \rangle \in E$ for some $n' \in N$ and $c = \lambda(n)$;
- $n_0 = root(G_r)$, i.e., the initial node of G_r.

From σ_f we can derive an actual perfect-recall strategy $\sigma : Obs^* \to Act_a$ as follows. We extend the deterministic function *succ* to observation histories $h \in Obs^*$ of length ℓ in the obvious manner. Then we define σ as the function: $\sigma(h) = act(n, c, last(h))$, where $n = succ(root(G_r), h^{n-1})$, $h^{\ell-1}$ is the prefix of h of length ℓ-1 and $c = \lambda(n)$ is the current configuration. Notice that such strategy is a partial function, dependent on the environment set $\mathcal{E}$: it is only defined for observation histories embodied by the run r.

It can be show that the procedure above, based on the algorithm UNPACK, is correct, in the sense that the executions of the strategy it produces are the same as those of the strategy generated by the automaton constructed for Theorem 11.

Example 6*:* Let us consider again the three environments, the agent and goal as in Example 4. Several strategies do exist. In particular, an accepting run r for $\mathcal{A}_{\mathcal{P}}$ is depicted in Figure 9, from which a strategy σ can be unpacked. Strategy σ can be equivalently represented as in Figure 10 as a function from observation histories to actions. For instance, $\sigma(c_0, \{\texttt{posA}, \texttt{nil}, \texttt{danger}, \texttt{nil}\}) = goE$. In particular, being all environments indistinguishable in the initial state (the agent receives the same observation `posA`), this strategy prescribes action *goE* for the three of them. Resulting states are such that both Env_1 and Env_3 provide perception `nil`, whereas Env_2 provides perception `danger`. Having received different observation histories so far, strategy σ is allowed to select different action for Env_2: *goE* for Env_2 and *wait* for Env_1 and Env_3. In fact, according to protocol *poss*, action *wait* is not an option for Env_2, whereas action *goE* is not significant for Env_3, though it avoids an accident in Env_1. In this example, by executing the strategy, the agent eventually receives different observation histories from each environment, but this does not necessary hold in general: different environments could also remain indistinguishable forever. ■

There is still no link between synthesized strategies and agents. The main idea is that a strategy can be easily seen as a sort of an agents' protocol refinement where the states used by the agents are extended to store the (part of the) relevant history. This is done in the next section.

7.6 Embedding Strategies into Protocols

We have seen how it is possible to synthesize perfect-recall strategies that are function of the observation history the agent received from the environment. Computing such strategies in general results into a function that requires an unbounded amount of memory. Nonetheless, the technique used to solve the problem shows that (*i*) if a strategy does exist, there exists a bound on the information actually required to compute and execute it and (*ii*) such strategies are finite-state. More precisely, from the run satisfying the LTL specification, it is possible to build the finite-state strategy $\sigma_f = \langle N, succ, act, n_0\rangle$. We now incorporate such a finite-state strategy into the agent protocol, by suitably expanding the configuration space of the agent to store in the configuration information needed to execute the finite state strategy. This amounts to define a new configuration space $\overline{Conf} = Conf \times N$ (hence a new local state space $\overline{L}$).

Formally, given the original agent $Ag = \langle Conf, Act_a, Obs, L, poss, \delta_a, c_0\rangle$ and the finite state strategy $\sigma_f = \langle N, succ, act, n_0\rangle$, we construct a new agent $\overline{Ag} = \langle \overline{Conf}, Act_a, Obs, \overline{L}, \overline{poss}, \overline{\delta}_a, \overline{c}_0\rangle$ where :

- Act_a and Obs are as in Ag;
- $\overline{Conf} = Conf \times N$ is the new set of configurations;
- $\overline{L} = \overline{Conf} \times Obs$ is the new local state space;

- $\overline{poss} : \overline{L} \to Act_a$ is an action-deterministic protocol defined as:

$$\overline{poss}(\langle c, n\rangle, o) = \begin{cases} \{\alpha\}, & \text{iff } act(n, c, o) = \alpha \\ \emptyset, & \text{if } act(n, c, o) \text{ is undefined;} \end{cases}$$

- $\overline{\delta}_a : \overline{L} \times Act_a \to \overline{Conf}$ is the transition function, defined as:

$$\overline{\delta}_a(\langle\langle c, n\rangle, o\rangle, a) = \langle \delta_a(c, o), succ(n, o, a)\rangle;$$

- $\overline{c_0} = \langle c_0, n_0\rangle$.

On this new agent the original strategy can be phrased as a state-base strategy:

$$\overline{\sigma} : \overline{Conf} \times Obs \to Act_a$$

simply defined as: $\overline{\sigma}(\langle c, n\rangle, o) = \overline{poss}(\langle c, n\rangle, o)$.

It remains to understand in what sense we can think the agent $\overline{Ag}$ as a refinement or customization of the agent Ag. To do so we need to show that the executions allowed by the new protocol are also allowed by the original protocol, in spite of the fact that the configuration spaces of the two agents are different. We show this by relaying on the theoretical notion of *simulation*, which formally captures the ability of one agent (Ag) to simulate, i.e., copy move by move, another agent ($\overline{Ag}$).

Given the two agents Ag_1 and Ag_2, a *simulation relation* is a relation $\mathcal{S} \subseteq L_1 \times L_2$ such that $\langle l_1, l_2\rangle \in \mathcal{S}$ implies that:

> if $l_2 \xrightarrow{\alpha} l_2'$ and $\alpha \in poss_2(l_2)$ then there exists l_1' such that $l_1 \xrightarrow{\alpha} l_1'$ and $\alpha \in poss_1(l_1)$ and $\langle l_1', l_2'\rangle \in \mathcal{S}$.

where $l_i \xrightarrow{\alpha} l_i'$ iff c_i' is the agent configuration in l_i' and $c_i' = \delta_a(l_i, \alpha)$. We say that agent Ag_1 *simulates* agent Ag_2 iff there exists a simulation relation $\mathcal{S}$ such that $\langle l_1^0, l_0^2\rangle \in \mathcal{S}$ for each couple of initial local states $\langle l_1^0, l_2^0\rangle$ with the same initial observation.

Theorem 13. *Agent Ag simulates* $\overline{Ag}$.

Proof. First, we notice that $\overline{poss}(\langle c, n\rangle, o) \subseteq poss(\langle c, o\rangle)$ for any $c \in Conf, o \in Obs$. In fact, since we are only considering allowed strategies, the resulting protocol $\overline{poss}$ is compatible with agent Ag. The result follows from the fact that original configurations are kept as fragment of both L and $\overline{L}$. Second, being both the agent and environments deterministic, the result of applying the same action α from states $\langle\langle c, n\rangle, o\rangle \in \overline{L}$ and $\langle c, o\rangle \in L$ are states $\langle\langle c', n'\rangle, o'\rangle$ and $\langle c', o'\rangle$, respectively.

Finally, assume towards contradiction that $\overline{Ag}$ is not simulated by Ag. This implies that there exists a sequence of length $n \geq 0$ of local states $l_0 \xrightarrow{\alpha_1} l_1 \xrightarrow{\alpha_2} \dots \xrightarrow{\alpha_k} l_k$ of $\overline{Ag}$, where l_0 is some initial local state, and a corresponding sequence $\overline{l}_0 \xrightarrow{\alpha_1} \overline{l}_1 \xrightarrow{\alpha_2} \dots \xrightarrow{\alpha_k} \overline{l}_k$ of $\overline{Ag}$, starting from a local state $\overline{l}_0$ sharing the same observation of l_0, such that $\alpha \in \overline{poss}(\overline{l}_k)$ but $\alpha \notin poss(l_k)$ for some α. For what observed before, l_k and $\overline{l}_k$ share the same agent's configuration; in particular, they are of the form $\overline{l}_k = \langle\langle c_k, n_k\rangle, o_k\rangle$ and $l_k = \langle c_k, o_k\rangle$. Hence $\overline{poss}(\langle\langle c_k, n_k\rangle, o_k\rangle) \subseteq poss(\langle c_k, o_k\rangle)$ and we get a contradiction. □

Theorem 14. $\overline{Ag}$ *is nonblocking.*

It follows from the fact that a strategy σ that is a solution for problem $\mathcal{P}$ is a prefix-closed function and it is allowed by Ag. Hence, for any $\bar{l} \in \overline{L}$ reachable from any initial local state by applying σ, we have $\overline{poss}(\bar{l}) \neq \emptyset$.

From Theorem 1 and results in previous sections we have:

Theorem 15. *Any execution of agent $\overline{Ag}$ over each environment Env_i satisfies the original agent specification Ag and the goal specification.*

7.7 A Notable Variant

Finally, we consider a variant of our problem where we specify LTL goals directly in terms of states of each environment in the environment set. In other words, instead of having a single goal specified over the percepts of the environments we have one LTL goal for each environment. More precisely, as before, we assume that a single strategy has to work on the whole set of deterministic environments. As previously, we require that $Act_a \subseteq \bigcap_{i=1,\ldots,k} Act_{ei}$ and that all environment share the same alphabet of perceptions $Perc$. Differently from before, we associate a distinct goal to each environment. We take as input alphabet of each goal specification $\mathcal{G}_i$ the set of environment's state E_i, i.e., $\mathcal{G}_i = \langle E_i, G_i, g_{i0}, \gamma_i, G_i^{acc} \rangle$. All goals are thus intimately different, as they are strictly related to the specific environment. Intuitively, we require that a strategy for agent Ag satisfies, in all environments Env_i, its corresponding goal $\mathcal{G}_i$. In other words, σ is a solution of the generalized planning problem $\mathcal{P} = \langle Ag, \mathcal{E}, \mathcal{G} \rangle$ iff it is allowed by Ag and it generates, for each environment Env_i, an infinite trace τ_i that is accepted by $\mathcal{G}_i$.

Devising perfect-recall strategies now requires only minimal changes to our original automata-based technique to take into account that we have now k distinct goals one for each environment. Given Ag and $\mathcal{E}$ as defined before, and k goals $\mathcal{G}_i = \langle E_i, G_i, g_{i0}, \gamma_i, G_i^{acc} \rangle$, we build the generalized Büchi automaton $\mathcal{A}_{\mathcal{P}} = \langle Act_a^k, W, w_0, \rho, W^{acc} \rangle$ as follows. Notice that each automaton $\mathcal{G}_i$ has E_i as input alphabet.

- $Act_a^k = (Act_a)^k$ is the set of k-vectors of actions;
- $W = E^k \times L^k \times G_1 \times \ldots \times G_k \times \wp(\equiv)$;
- $w_0 = \langle e_{i0}, \ldots, e_{k0}, c_{i0}, \ldots c_{k0}, g_{i0}, \ldots, g_{k0}, \equiv_0 \rangle$ where

  $i \equiv_0 j$ iff $obs(perc_i(e_{i0})) = obs(perc_j(e_{j0}))$;
- $\langle \vec{e}', \vec{c}', \vec{g}', \equiv' \rangle \in \rho(\langle \vec{e}, \vec{c}, \vec{g}, \equiv \rangle, \vec{\alpha})$ iff
  - if $i \equiv j$ then $\alpha_i = \alpha_j$;
  - $e_i' = \delta_i(e_i, \alpha_i)$;
  - $c_i' = \delta_a(l_i, \alpha_i) \wedge \alpha_i \in poss(l_i)$
    with $l_i = \langle c_i, obs(perc_i(e_i)) \rangle$;
  - $g_i' = \gamma_i(g_i, e_i)$;
  - $i \equiv' j$ iff $i \equiv j \wedge obs(perc_i(e_i')) = obs(perc_j(e_j'))$.
- $W^{acc} = \{$
  $E^k \times Conf^k \times G_1^{acc} \times G_2 \times \ldots \times G_k \times \equiv$, ... ,
  $E^k \times Conf^k \times G_1 \times \ldots \times G_{k-1} \times G_k^{acc} \times \equiv \}$

The resulting automaton is similar to the previous one, and the same synthesis procedures apply, including the embedding of the strategy into an agent protocol. We also get the analogous soundness and completeness result and complexity characterization as before.

Example 7*:* Consider again Example 4 but now we require, instead of having a single goal $\phi_{\mathcal{G}}$, three distinct goals over the three environments. In particular for the car-train environments Env_1 and Env_2, we adopt the same kind of goal as before, but avoiding certain cells for environments, e.g., $\phi_{\mathcal{G}_1} = (\Box\Diamond e_{11}) \wedge (\Box\Diamond e_{24}) \wedge \Box\neg(\langle c_{13}, c_{13}\rangle \vee \langle c_{23}, c_{23}\rangle) \wedge \Box\neg\langle e_{22}, e_t\rangle$ and $\phi_{\mathcal{G}_2} = (\Box\Diamond e_{21}) \wedge (\Box\Diamond e_{24}) \wedge \Box\neg(\langle c_{23}, c_{23}\rangle \vee \ldots \vee \langle c_{14}, c_{14}\rangle) \wedge \Box\neg\langle e_{11}, e_{14}\rangle$, whereas in the Wumpus world we only require to reach the gold after visiting initial position: $\phi_{\mathcal{G}_3} = \Box(e_{11} \rightarrow \Diamond e_{34}) \wedge \Box\neg(c_{23} \vee c_{31})$. It can be shown that a (perfect-recall) strategy for achieving such goals exists. In fact, there exists at least one strategy (e.g., one extending the prefix depicted in Figure 10 avoiding in Env_1 and Env_2 states mentioned above) that satisfies goal $\phi_{\mathcal{G}}$ over all environments as well as these three new goals (in particular, if a strategy satisfies $\phi_{\mathcal{G}}$ then it satisfies $\phi_{\mathcal{G}_3}$ too). Such a strategy can be transformed into an agent protocol, by enlarging the configuration space of the agent, as discussed in the previous section.

7.8 Discussion

In this section we investigated the synthesis of an agent's protocol to satisfy LTL specifications while dealing with multiple, partially-observable environments. In addition to the computationally optimal procedure here introduced, we explored an automata-based protocol refinement for a perfect-recall strategy that requires only finite states.

There are several lines we wish to pursue in the future. Firstly, we would like to implement the procedure here described and benchmark the results obtained in explicit and symbolic settings against planning problems from the literature. We note that current model checkers such as MCMAS [LQR09a] and MCK [GvdM04] support interpreted systems, the semantics here employed.

It is also of interest to explore whether the procedures here discussed can be adapted to other agent-inspired logics, such as epistemic logic [RFV95]. Epistemic planning [vdHW02], i.e., planning for epistemic goals, has been previously discussed in the agents-literature before, but synthesis methodologies have not, to our knowledge, been used in this context.

When dealing with LTL goals we need to consider that the agent cannot really monitor the achievement of the specification. Indeed every linear temporal specification can be split into a "liveness" part which can be checked only considering the entire run and a "safety" part that can be checked on finite prefixes of such runs [BK08]. Obviously the agent can look only at the finite history of observations it got so far, so being aware of achievement of LTL properties is quite problematic in general. This issue is related to runtime verification and monitoring [EFH$^+$03, BLS11], and in the context of AI, it makes particularly attractive to include in the specification of the dynamic property aspects related to the knowledge that the agent acquires, as allowed by interpreted systems.

8 Conclusions

In this document we have described the contributions at the conclusion of WP2 of the ACSI project. We have identified the project requirements concerning the main topic of WP2, i.e., verification and synthesis, and have entirely fulfilled these requirements.

Our achievements include:

- introduction of a formal model and a specification language for reasoning about temporal-epistemic properties of artifact systems, over which we developed an abstraction methodology that guarantees decidability in some cases of practical interests (Requirements **R1**-**R6**);
- development of an actual technique for detecting when the executions of an artifact system are guaranteed to manipulate only a finite set of new elements, which opens the possibility of adapting standard Model Checking techniques for general (i.e., linear- and branching-time) temporal properties, to artifact systems (Requirements **R3**, **R5** and **R7**);
- relevant results in synthesis techniques for artifact-centric systems (Requirements **R8**). Specifically, we presented a sound and complete procedure for synthesizing agent protocols satisfying LTL specifications for multiple, partially-observable environments.

These contributions represent significant advances in our understanding of the verification and synthesis problems in the context of artifact systems. As we already discussed, at this stage, the achievements in verification constitute a basis also for synthesis as, similarly to the propositional case, most of the framework developed for verification can be used (possibly extended) for synthesis purposes.

The main difficulty we encountered, which is also the characterising feature of artifact systems from the verification viewpoint, is the presence of data, its infiniteness, and the relevance of properties over them. These facets rule out existing approaches, e.g., propositional model checking, and call for a suitable first-order generalisation. Therefore in our results, we naturally extend methodologies and techniques developed in the broad area of formal verification and synthesis, including abstraction and verification of database-driven systems [Via09].

All results were achieved in line with the original plan as published in the DOW.

The work presented in Section 4 and 5 constitutes a theoretical framework for reasoning about temporal-epistemic properties of artifact systems, as well as for the verification of GSM models. Based on this, actual verification techniques are being developed. In particular, we expect to be able to verify properties similar to those addressed in Section 6, extended with epistemic modalities. Our assumption will be that the artifact system does not exceed a size bound fixed *a priori*. We note that this is a different condition than the one required in the method presented in Section 6. Indeed, the technique presented in Section 6 requires that the number of different values manipulated during each execution is finite, whereas we are interested in bounding the number of different values present at each state, although along an execution the number of manipulated values can be infinite.

As to the contribution described in Section 6, an important issue is the development of *effective* verification techniques. As a matter of fact, although decidable, the verification task turns out extremely complex, namely time-exponential. Thus, there is the need for developing techniques, possibly based on abstraction (used to narrow the search space), that perform well in the practice. Another relevant point concerns richness of the specification language. In particular, verifiable properties do not include the equality symbol, and the use of quantifiers is restricted to local states or to the initial state (possibly combined), while interactions among generic states is not in general possible. Weakening these restrictions would allow to capture a larger class of properties over artifact systems, and is certainly of interest to the project. Finally, the work currently assumes that no artifact is created or deleted during execution. While this is not as restrictive as it may appear (in fact, an artifact instance corresponds to a whole database), it certainly constitutes a limitation on the class of systems that can be verified. Weakening this restriction requires a major extension of the introduced framework.

9 Appendix

Lemma 1 Assume that $\mathcal{P}'$ simulates $\mathcal{P}$. For any FO-$\forall$ACTLK formula $\phi \in \mathcal{L}'$, if $\mathcal{P}' \models \phi$ then $\mathcal{P} \models \phi$.

Proof. Assume a simulation pair $(\simeq, \approx)$ between $\mathcal{P}$ and $\mathcal{P}'$. By induction on $\phi \in \mathcal{L}'$ we show that

$$\text{if } (\mathcal{P}'^{\sigma'}, s') \models \phi, s \simeq s' \text{ and } \sigma(x) \approx \sigma'(x) \text{ then } (\mathcal{P}^{\sigma}, s) \models \phi \tag{1}$$

First we remark that for all terms $t \in \mathcal{L}'$, if $\sigma(x) \approx \sigma'(x)$ then $I^{\sigma}(t) \approx I'^{\sigma'}(t)$ because of clause (f) in Def. 4.12. Now, the base case for ϕ equal to $P^n(\vec{t})$ or $\neg P^n(\vec{t})$ follows from (e) and the previous remark. For instance, $(\mathcal{P}'^{\sigma'}, s') \models P^n(\vec{t})$ iff $I'^{\sigma'}(\vec{t}) \in I'(P^n, s')$. But $I^{\sigma}(\vec{t}) \approx I'^{\sigma'}(\vec{t})$ and by (e) it follows that $I^{\sigma}(\vec{t}) \in I(P^n, s)$, that is, $(\mathcal{P}^{\sigma}, s) \models P^n(\vec{t})$. The induction step for CTL and epistemic modalities follows from (c) and (d) respectively. See [CDLR09] for details. As to the universal quantifier, if $(\mathcal{P}'^{\sigma'}, s') \models \forall x \psi$, $s \simeq s'$ and $\sigma(x) \approx \sigma'(x)$, then for all $a' \in U'$, $(\mathcal{P}'^{\sigma'^{a'}_x}, s') \models \psi$. Now consider all the $a^* \in U'$ such that there exists $a \in U$ and $a \approx a^*$. Notice that $\sigma^a_x \approx \sigma'^{a^*}_x$ and by induction hypothesis $(\mathcal{P}^{\sigma^a_x}, s) \models \psi$. By (b) it follows that $(\mathcal{P}^{\sigma}, s) \models \forall x \psi$. Finally, the lemma follows from (1) and simulation requirements (a) and (b). □

Lemma 2 Assume that $\mathcal{P}'$ simulates$^+$ $\mathcal{P}$. For any FO-ACTLK formula $\phi \in \mathcal{L}'$, if $\mathcal{P}' \models \phi$ then $\mathcal{P} \models \phi$.

Proof. Assume a simulation$^+$ pair $(\simeq, \approx)$ between $\mathcal{P}$ and $\mathcal{P}'$. By induction on $\phi \in \mathcal{L}'$ we show that (1) holds also in this case. The proof goes as in Lemma 1, but for the case of the existential quantifier: if $(\mathcal{P}'^{\sigma'}, s') \models \exists x \psi$, $s \simeq s'$ and $\sigma(x) \approx \sigma'(x)$, then there exists $a' \in U'$ such that $(\mathcal{P}'^{\sigma'^{a'}_x}, s') \models \psi$. Now consider $a \in U$ such that $a \approx a'$, which exists by clause (b) in Def. 4.13. Notice that $\sigma^a_x \approx \sigma'^{a'}_x$ and by induction hypothesis $(\mathcal{P}^{\sigma^a_x}, s) \models \psi$. Thus, $(\mathcal{P}^{\sigma}, s) \models \exists x \psi$. □

□

Lemma 3 If $\mathcal{P}'$ is a quotient of $\mathcal{P}$, then $\mathcal{P}'$ simulates$^+$ $\mathcal{P}$.

Proof. We show that the relations $\simeq\, = \{\langle s, [s]\rangle \mid s \in U\}$ and $\approx\, = \{\langle a, [a]\rangle \mid a \in U\}$ are a simulation$^+$ pair for $\mathcal{P}$ and $\mathcal{P}'$. Simulation requirements (a) and (b) follow from 6 and 1 in Def. 4.14 respectively. Requirement (c) follows from 4 and 5; while requirement (d) follows by the definition of equivalence classes. Simulation requirements (e) and (f) follow from 7 and 8 respectively. Finally, the simulation$^+$ requirements (b) in Def. 4.13 trivially hold. □

Lemma 4 The abstract model $\mathcal{M}'$ simulates$^+$ $\mathcal{P}$.

Proof. The proof consists in showing that the relations $\simeq\, = \{\langle s, [s]\rangle \mid s \in \mathcal{S}\}$ and $\approx\, = \{\langle a, [a]_s\rangle \mid a \in U, s \in \mathcal{S}\}$ are a simulation$^+$ pair for $\mathcal{P}$ and $\mathcal{M}'$, and it is similar to Lemma 3.

Simulation requirements (a) and (b) follow by the definition of $\mathcal{S}'$ and U' in the abstract model. For (c) assume $g \to s$. By Def. 4.18, $[g] \to' [s]$. Thus,

$$g \to s \quad \text{implies} \quad [g] \to' [s] \tag{2}$$

from which (c) follows. For simulation requirement (d) assume that $g \sim_i s$ for $g, s \in \mathcal{S}$. By definition of QIS, $g_i = s_i$, that is , $[g_i] = [s_i]$, i.e.,

$$[g]_i = [s]_i \tag{3}$$

By (2) and simulation requirement (a), $[g], [s] \in \mathcal{S}'$. Thus, simulation requirement (d) follows by (3). Simulation requirements (e) and (f) follow from the definition of I' in Def. 4.18. Finally, the simulation is actually a simulation$^+$ as for every equivalence class $[a] \in \mathcal{M}'$ there exists $a \in \mathcal{P}$ such that $a \approx [a]$. $\square$

Theorem 1 The model checking problem for AC-MAS w.r.t. FO-CTLK is undecidable.

Proof. The theorem can be proved by showing that every Turing machine T whose tape contains an initial input I can be simulated by an artifact system $\mathcal{P}_{T,I}$, and that 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 Theorem 4.10 of [DSV07]. $\square$

Proposition 4.22 Given two isomorphic states s and s', an FO-formula φ, and two assignments σ and σ' equivalent for φ, we have that

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

Proof. The proof is by induction on the structure of φ. Consider the base case for the atomic formula $\varphi \equiv P(t_1, \ldots, t_k)$, then $(D_s, \sigma) \models \varphi$ iff $\langle \sigma(t_1), \ldots, \sigma(t_k) \rangle \in D_s(P)$. Since σ and σ' are equivalent for φ, and $s \simeq s'$, this is the case iff $\langle \sigma'(t_1), \ldots, \sigma'(t_k) \rangle \in D_{s'}(P)$, that is, $(D_{s'}, \sigma') \models \varphi$. The base case for $\varphi \equiv t = t'$ is proved similarly by observing that γ is a bijection, thus all (in)equalities between values assigned by σ and σ' are preserved. The inductive step for propositional connectives is straightforward. Finally, if $\varphi \equiv \forall y \psi$, then $(D_s, \sigma) \models \varphi$ iff for all $u \in adom(s)$, $(D_s, \sigma\binom{y}{u}) \models \psi$. Now consider the witness $\iota = \gamma|_{adom(s) \cup C}$ for $s \simeq s'$, where γ is as in Def.4.21. We have that $\sigma\binom{y}{u}$ and $\sigma'\binom{y}{\iota(u)}$ are equivalent for φ. By induction hypothesis $(D_s, \sigma\binom{y}{u}) \models \psi$ iff $(D_{s'}, \sigma'\binom{y}{\iota(u)}) \models \psi$. Since ι is a bijection, this is the case iff $(D_{s'}, \sigma') \models \varphi$. $\square$

Proposition 4.26 If an AC-MAS $\mathcal{P}$ satisfies req. 1 in Def. 4.25 and $adom(s_0) \subseteq C$, then req. 2 is also satisfied.

Proof. If $s \oplus t \simeq s' \oplus t'$, then there is a witness $\iota : adom(s) \cup adom(t) \cup C \mapsto adom(s') \cup adom(t') \cup C$ that is the identity on C, hence also on $adom(s_0)$. Since $s \sim_i t$, then $l_i(s) = l_i(t)$, and also $l_i(s') = \iota(l_i(s)) = \iota(l_i(t)) = l_i(t')$. 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 $s_0 \to s_1 \to \ldots \to s_k$ s.t. $s_k = t$. Extend now ι to a total and injective function $\iota' : adom(s_0) \cup \cdots \cup adom(s_k) \cup C \mapsto U$. This can always be done by defining, for instance, $\iota'(u) = \iota(u)$ whenever $u \in adom(s) \cup adom(t) \cup C$, and $\iota'(u) = u$ otherwise. Now consider the sequence $\iota'(s_0), \iota'(s_1), \ldots, \iota'(s_k)$. Since $ad(s_0) \subseteq C$, we have that $\iota'(s_0) = s_0$. Further, $\iota'(s_k) = \iota(t) = t'$. By repeated applications of req. 1 we can show that $\iota'(s_{m+1}) \in \tau(\iota'(s_m), \alpha(\iota'(\vec{u})))$ whenever $s_{m+1} \in \tau(s_m, \alpha(\vec{u}))$ for $m < k$. Hence, the sequence is actually a run from s_0 to t'. Thus, $t' \in \mathcal{S}$, and $t \sim_i t'$. $\square$

Proposition 4.27 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 relation is sufficient to prove it is a bisimulation.

As regards (2) in Def. 4.23, let $\langle s, s' \rangle \in B$, and assume $s \to t$ 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 $\iota' : adom(s) \cup adom(t) \cup C \mapsto U$, for instance by assuming that $\iota'(u) = u$ whenever $u \notin adom(s) \cup adom(t) \cup C$. Then, by defining $t' = \iota'(t)$, 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, $s' \to t'$. Moreover, ι' is a witness for $t \simeq t'$, thus $\langle t, t' \rangle \in B$.

As to (3), if $\langle s, s' \rangle \in B$ 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 $\langle t, t' \rangle \in B$. □

Proposition 4.28 Consider two bisimilar and uniform AC-MAS $\mathcal{P}$ and $\mathcal{P}'$, two isomorphic states $s \in \mathcal{P}$ and $s' \in \mathcal{P}'$, 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, if $s \to t$ and $|U'| \geq |adom(s) \cup adom(t) \cup C \cup \sigma(free(\varphi))|$, then there exists t' 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, if $s \sim_i t$ and $|U'| \geq |adom(s) \cup adom(t) \cup C \cup \sigma(free(\varphi))|$, then there exists t' 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 simulation there exists a $t'' \in \mathcal{S}'$ s.t. $s' \to t''$, $s \oplus t \simeq s' \oplus t''$, and $t \approx t''$. We define a function j s.t. $Dom(j) \doteq adom(s) \cup adom(t) \cup C$, and we partition it into:

- $Dom(\gamma) \doteq adom(s) \cup C \cup (adom(t) \cap \sigma(free(\varphi))$;
- $Dom(\iota') \doteq adom(t) \setminus Dom(\gamma)$;

Then we define $j : Dom(j) \mapsto U_2$ as follows:

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

where $\iota' : Dom(\iota') \mapsto U' \setminus Im(\gamma)$ is any invertible (total) function. It can be proved that $|U'| \geq |adom(s) \cup adom(t) \cup C \cup \sigma(free(\varphi))|$ implies $|U' \setminus Im(\gamma)| \geq |Dom(\iota')|$, thus such ι' exists. To see the former fact observe that $|Im(\gamma)| = |adom(s') \cup C \cup \sigma'(free(\varphi))| = |adom(s) \cup C \cup \sigma(free(\varphi))|$.

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, $s' \oplus t' \simeq s' \oplus t''$. Finally, $s' \to t'$ 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. 4.27 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) is similar to (1), and is omitted. □

Proposition 4.29 Consider two bisimilar uniform AC-MAS $\mathcal{P}$ and $\mathcal{P}'$, two isomorphic states $s \in \mathcal{P}$ and $s' \in \mathcal{P}'$, 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, if $r(0) = s$ and for all $i \geq 0$, $|U'| \geq |adom(r(i)) \cup adom(r(i+1)) \cup C \cup \sigma(free(\varphi))|$, then there exists a t.e. run r' 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 $r'(i)$.

Proof. Let r be a t.e. run s.t. $|U'| \geq |adom(r(i)) \cup adom(r(i+1)) \cup C \cup \sigma(free(\varphi))|$ for all $i \geq 0$. We inductively build r'. For $i = 0$, let $r'(0) = s'$. By hypothesis, r is s.t. $|U'| \geq |adom(r(0)) \cup adom(r(1)) \cup C \cup \sigma(free(\varphi))|$. Thus, since $r(0) \rightsquigarrow r(1)$, by Prop. 4.28 there exists t' s.t. $r'(0) \rightsquigarrow t'$, σ and σ' are equivalent for φ w.r.t. $r(1)$ and t'. Let $r'(1) = t'$.

The case for $i > 0$ is similar. Assume that $r(i) \approx r'(i)$ and σ and σ' are equivalent for φ w.r.t. $r(i)$ and $r'(i)$. Since $r(i) \rightsquigarrow r(i+1)$ and $|U'| \geq |adom(r(i)) \cup adom(r(i+1)) \cup C \cup \sigma(free(\varphi))|$, by Prop. 4.28 there exists t' s.t. $r'(i) \rightsquigarrow t'$, $r(i+1) \approx t'$, and σ and σ' are equivalent for φ w.r.t. $r(i+1)$ and t'. Let $r'(i+1) = t'$. It is clear that r' is a t.e. run in $\mathcal{P}'$. □

Lemma 7 Consider two bisimilar and uniform AC-MAS $\mathcal{P}$ and $\mathcal{P}'$, two isomorphic states $s \in \mathcal{P}$ and $s' \in \mathcal{P}'$, 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 $|U'| \geq |adom(r(k)) \cup adom(r(k+1)) \cup C \cup \sigma(free(\varphi))| + |vars(\varphi) \setminus free(\varphi)|$;
2. for every t.e. run r' s.t. $r'(0) = s'$, for all $k \geq 0$ we have $|U| \geq |adom(r'(k)) \cup adom(r'(k+1)) \cup C \cup \sigma'(free(\varphi))| + |vars(\varphi) \setminus free(\varphi)|$;

then

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

Proof. The proof is by induction on the structure of φ, by using Prop. 4.29. We prove the $\Rightarrow$ part. The $\Leftarrow$ part is proved analogously by swapping indexes, as the hypotheses are symmetric. The base case for atomic formulas follows from Prop. 4.22. The inductive cases for propositional connectives are straightforward.

For $\varphi \equiv \forall x \psi$, assume that $x \in free(\psi)$ (otherwise consider ψ), 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. By noticing that $const(\psi) = const(\varphi)$ and $free(\psi) = free(\varphi) \cup \{x\}$, it is apparent that $\sigma\binom{x}{u}$ 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 $\sigma(free(\varphi))$. Similarly for σ'. Further, $|vars(\psi) \setminus free(\psi)| = |vars(\varphi) \setminus free(\varphi)| - 1$, as $vars(\psi) = vars(\varphi)$, $free(\psi) = free(\varphi) \cup \{x\}$, and $x \notin free(\varphi)$. This enables the application of the induction hypothesis, thus obtaining that $(\mathcal{P}, s, \sigma\binom{x}{u}) \models \psi$ iff $(\mathcal{P}', s', \sigma'\binom{x}{\gamma(u)}) \models \psi$. Since γ is a bijection, the result easily follows.

For $\varphi \equiv AX\psi$, assume for contradiction that $(\mathcal{P}, s, \sigma) \models \varphi$ and $(\mathcal{P}', s', \sigma') \not\models \varphi$. Then, there exists a run r' s.t. $r'(0) = s'$ and $(\mathcal{P}', r'(1), \sigma') \not\models \psi$. By Prop. 4.29 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 also satisfies req. 1. Moreover, the same requirement is necessarily satisfied by all those t.e. runs r'' s.t. $r''(0) = r(i)$, for some $i \geq 0$ (otherwise, there would exist some t.e. run from s not satisfying the requirement), and the same occurs with req. 2, for all 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 $vars(\varphi) = vars(\psi)$ and $free(\varphi) = free(\psi)$). But then we obtain $(\mathcal{P}, r(1), \sigma) \not\models \psi$, thus $(\mathcal{P}, r(0), \sigma) \not\models AX\psi$. 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 quantified variables). Let r be a run s.t. $r(0) = s$, then there exists $k \geq 0$ s.t. $(\mathcal{P}, r(k), \sigma) \models \phi$, and for every j, $0 \leq j < k$ implies $(\mathcal{P}, r(j), \sigma) \models \psi$. By Prop. 4.29 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 $\gamma_{i,\psi} = \gamma_i|_{ad(r(i))\cup C\cup\sigma(free(\psi))}$ and $\gamma_{i,\phi} = \gamma_i|_{ad(r(i))\cup C\cup\sigma(free(\phi))}$. Since $free(\psi), free(\phi) \subseteq free(\varphi)$, 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, it can be seen that req. 1 holds for all t.e. runs r'' s.t. $r''(0) = r(i)$, for some $i \geq 0$, and req. 2 holds for all t.e. runs r''' s.t. $r'''(0) = r'(i)$. Now observe that $|\sigma(free(\phi))|, |\sigma(free(\psi))| \leq |\sigma(free(\varphi))|$ and $(vars(\varphi) \setminus free(\varphi)) = (vars(\psi) \setminus free(\psi)) \cup (vars(\phi) \setminus free(\phi))$, with $(vars(\psi) \setminus free(\psi)) \cap (vars(\phi) \setminus free(\phi)) = \emptyset$ by the above assumption on variables common to ψ and ϕ. Thus $|vars(\varphi) \setminus free(\varphi)| = |(vars(\psi) \setminus free(\psi)| + |(vars(\phi) \setminus free(\phi)|$. Hence $|(vars(\psi) \setminus free(\psi)|, |(vars(\phi) \setminus free(\phi)| \leq |vars(\varphi) \setminus free(\varphi)|$. Therefore, if reqq. 1 and 2 hold, then they also hold if φ is uniformly replaced by 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 $(\mathcal{P}', r'(i), \sigma') \models \phi$. Finally, 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 for contradiction that $(\mathcal{P}, s, \sigma) \models \varphi$ and $(\mathcal{P}', s', \sigma') \not\models \varphi$. 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 Prop. 4.29 there exists a t.e. 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 the only variables common to ψ and ϕ occur free in both formulas, it can be proved as in the case for $E\psi U\phi$ that the induction hypothesis applies for every $i \geq 0$, So, $(\mathcal{P}, r(i), \sigma) \models \psi$ 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 $\varphi \equiv K_i\psi$, we assume for contradiction that $(\mathcal{P}, s, \sigma) \models \varphi$ and $(\mathcal{P}', s', \sigma') \not\models \varphi$. Then, there exists a s'' s.t. $s' \sim_i s''$ and $(\mathcal{P}', s'', \sigma') \not\models \psi$. By Prop. 4.29 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, we can apply the induction hypothesis, and we obtain that $(\mathcal{P}, s''', \sigma) \not\models \psi$. Hence $(\mathcal{P}, s, \sigma) \not\models K_i\psi$, which is a contradiction.
□

Theorem 4 Consider two bisimilar and uniform AC-MAS $\mathcal{P}$ and $\mathcal{P}'$, and an FO-CTLK formula φ. If

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

then

$$\mathcal{P} \models \varphi \quad \text{iff} \quad \mathcal{P}' \models \varphi$$

Proof. Equivalently, we prove that if $(\mathcal{P}, s_0, \sigma) \not\models \varphi$ for some σ, then there is σ' such that $(\mathcal{P}', s'_0, \sigma') \not\models \varphi$, and viceversa. Firstly, observe that hypotheses 1 and 2 imply, respectively, hypotheses 1 and 2 of Lemma 7. Secondly, 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 Lemma 7 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 is proved similarly. □

Lemma 8 The abstraction $\mathcal{P}'$ of a uniform AC-MAS $\mathcal{P}$ is uniform.

Proof. The result follows by the definition of τ' in $\mathcal{P}'$ and uniformity of $\mathcal{P}$. Consider $s, t, s' \in \mathcal{S}'$, $t' \in \mathcal{D}(U')$, and $\alpha \in Act'$ s.t. $t \in \tau'(s, \alpha(\vec{u}))$ for some $\vec{u} \in U'^n$, and $s \oplus t \simeq s' \oplus t'$ for some witness ζ. By definition of τ' there are s'', t'', and $\vec{u}' \in U^n$ s.t. $t'' \in \tau(s'', \alpha(\vec{u}'))$, $s'' \oplus t'' \simeq s \oplus t$ for some witness ι, and $\vec{u} = \iota(\vec{u}')$. Hence, $s'' \oplus t'' = \iota^{-1}(s) \oplus \iota^{-1}(t) \simeq \iota^{-1}(s') \oplus \iota^{-1}(t')$ for the witness $\iota^{-1} \circ \zeta \circ \iota$. Since $\mathcal{P}$ is uniform, $\iota^{-1}(t') \in \tau(\iota^{-1}(s'), \alpha(\iota^{-1} \circ \zeta(\vec{u}))))$. Again by definition of $\mathcal{P}'$ this means that $t' \in \tau(s', \alpha(\zeta(\vec{u})))$. Thus, $\mathcal{P}'$ is uniform. □

Theorem 5 Given an infinite, b-bounded and uniform AC-MAS $\mathcal{P}$, an FO-CTLK formula φ, and a finite set $U' \supseteq C$ s.t. $|U'| \geq 2b + |C| + \max\{vars(\varphi), N_{Ag}\}$, the abstraction $\mathcal{P}'$ is finite, uniform and bisimilar to $\mathcal{P}$. Morever,

$$\mathcal{P} \models \varphi \quad \text{iff} \quad \mathcal{P}' \models \varphi$$

Proof. The result follows from the fact that U' contains enough elements to simulate the transition relation in $\mathcal{P}$. We know that $\mathcal{P}'$ is finite, as U' is, and it is uniform by Lemma 8. We show that $\mathcal{P}'$ is bisimilar to $\mathcal{P}$. Consider $s \in \mathcal{P}$ and $s' \in \mathcal{P}'$ s.t. $s \simeq s'$, and assume that $s \to t$ for some $t \in \mathcal{P}$. We show that there exists $t' \in \mathcal{P}'$ s.t. $s' \to t'$ and $s \oplus t \simeq s' \oplus t'$. If $s \to t$ then there is $\alpha(\vec{u}) \in Act(U)$. Since $|U'| \geq 2b + |C| + N_{Ag}$, $\sum_{A_i \in Ag} |\vec{u}_i| \leq N_{Ag}$, and $|adom(s) \cup adom(s')| \leq 2b$, the isomorphism ι witnessing $s \simeq s'$ can be extended to an injective function $\iota' : adom(s) \cup adom(s') \cup C \cup \bigcup_{A_i \in Ag} \vec{u}_i \mapsto U'$. Now let $t' = \iota'(s')$. By the way ι' has been defined, it can be seen that $s \oplus t \simeq s' \oplus t'$. Further, by the definition of $\mathcal{P}'$ we have that $t' \in \tau'(s', \alpha(\vec{u}'))$ for $\vec{u}' = \iota'(\vec{u})$, that is, $s' \to t'$. Also, if $s \in \mathcal{S}$, $s' \in \mathcal{S}'$, $s \simeq s'$ and $s' \to t'$, then there exists $t \in \mathcal{S}$ s.t. $s \to t$ by definition of $\mathcal{P}'$. Thus, $\mathcal{P}$ and $\mathcal{P}'$ are bisimilar. Finally, notice that the assumptions in Theor. 4 are satisfied. In particular, for all t.e. run r s.t. $r(0) = s_0$, and for all $k \geq 0$, $|U'| \geq |adom(r(k)) \cup adom(r(k+1)) \cup C| + |vars(\varphi)|$. Hence, we obtain that $\mathcal{P} \models \varphi$ iff $\mathcal{P}' \models \varphi$. □

Theorem 7 The complexity of the model checking problem for finite AC-MAS w.r.t. the language FO-CTLK is PSPACE-complete.

Proof. This result is obtained by combining the complexity for model checking the first-order fragment of FO-CTLK and the temporal epistemic fragment. PSPACE-hardness follows by reduction to first-order model checking. To show that the problem is in PSPACE, we briefly describe an algorithm which works in NPSPACE. Since NPSPACE = PSPACE, the result follows. Given an AC-MAS $\mathcal{P}$ and an FO-CTLK formula φ, guess an assignment σ and check if $(\mathcal{P}, D_0, \sigma) \models \varphi$. This can be done according to the structure of φ. If φ is atomic, this check can be done in PSPACE. 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 main operator in φ is either a temporal or epistemic modality, then we can apply the algorithm to model check propositional CTLK, which is in P. As a result, the total complexity is in PSPACE. □

Lemma 9 For every $D, D' \in \mathcal{P}_\Gamma$, if $D \simeq D'$ for some witness ι, then $\Sigma_{D'} = \iota(\Sigma_D)$.

Proof. Notice that if ι is a witness for $D \simeq D'$, then in particular the attributes in Σ_D are mapped to the corresponding attributes in $\Sigma_{D'}$. Further, *Stg*, *Mst* and *Lcyc* remain the same. □

Lemma 10 For every D, D' , $D'' \in \mathcal{S}$, $D''' \in \mathcal{D}_\Gamma(U)$, if $D \oplus D' \simeq D'' \oplus D'''$ for some witness ι, then

$$(\Sigma_D, e, \Sigma_{D'}) \quad \text{implies} \quad (\Sigma_{D''}, \iota'(e), \Sigma_{D'''})$$

where ι' is any bijection extending ι

Proof. Assume that $(\Sigma_D, e, \Sigma_{D'})$ and ι is a witness for $D \oplus D' \simeq D'' \oplus D'''$. We show that $(\Sigma_{D''}, \iota'(e), \Sigma_{D'''})$ where ι' is a bijection extending ι. If $(\Sigma_D, e, \Sigma_{D'})$ then there is a sequence $\Sigma_0, \ldots, \Sigma_k$ of snapshots s.t. $\Sigma_0 = \Sigma_D$, $\Sigma_1 = \mathit{ImmEffect}(\Sigma_D, e)$, and $\Sigma_k = \Sigma_{D'}$. Also, for $1 \leq j \leq k-1$, Σ_{j+1} is obtained from Σ_j by the application of a PAC rule. We show that we can define a sequence $\Sigma'_0, \ldots, \Sigma'_k$ s.t. $\Sigma'_0 = \Sigma_{D''}$, $\Sigma'_1 = \mathit{ImmEffect}(\Sigma_{D''}, \iota'(e))$, $\Sigma'_k = \Sigma_{D'''}$, and for $1 \leq j \leq k-1$, Σ'_{j+1} is obtained from Σ'_j by the application of a PAC rule. This is sufficient to show that $(\Sigma_{D''}, \iota'(e), \Sigma_{D'''})$. First, for $0 \leq j \leq k$ define $\Sigma'_j = \iota'(\Sigma_j)$. By Lemma 9 we have that $\Sigma'_0 = \iota'(\Sigma_D) = \Sigma_{D''}$ and $\Sigma'_k = \iota'(\Sigma_{D'}) = \Sigma_{D'''}$. Also, it is clear that if $\Sigma_1 = \mathit{ImmEffect}(\Sigma_D, e)$, then $\Sigma'_1 = \iota'(\Sigma_1) = \iota'(\mathit{ImmEffect}(\Sigma_D, e))$ is equal to $\mathit{ImmEffect}(\iota'(\Sigma_D), \iota'(e)) = \mathit{ImmEffect}(\Sigma_{D'}, \iota'(e))$. Finally, we show that if Σ_{j+1} is obtained from Σ_j by an application of a PAC rule, then also Σ'_{j+1} is obtained from Σ'_j by a PAC rule. Consider the PAC rule $\langle \pi(x), \alpha(x), \gamma(x) \rangle$. We have that if $\Sigma_D \models \pi(\rho)$ for some artifact ID ρ in Σ_D, then clearly $\Sigma_{D'} \models \pi(\iota'(\rho))$. Further, $\Sigma_j \models \alpha(\rho) \equiv \chi(\rho) \wedge \psi(\rho)$, where $\chi(x)$ is an amenable sentry and $\psi(x)$ is of the form $\tau.m$, $\neg\tau.m$, $\tau.\mathit{active}_S$, or $\neg\tau.\mathit{active}_S$. Clearly, if $\Sigma_j \models \psi(\rho)$ then $\Sigma'_j \models \psi(\iota'(\rho))$. Further, since $\chi(x)$ is of the form **on** $\xi(x)$ **if** $\varphi(x)$ and $\varphi(x)$ is an FO-formula in $\mathcal{L}_{\mathcal{D}_\Gamma}$, then by Prop. 4.22 we have that $\Sigma'_j \models \chi(\iota'(\rho))$. Hence, $\Sigma'_j \models \alpha(\iota'(\rho))$. Finally, if Σ_{j+1} is constructed from Σ_j by applying the effect called for by $\gamma(\rho)$, then Σ'_{j+1} is constructed from Σ'_j by applying the effect called for by $\gamma(\iota'(\rho))$. Thus, we have the desired result. □

Theorem 8 If the GSM program Γ is amenable, then the AC-MAS $\mathcal{P}_\Gamma$ is uniform.

Proof. Let assume that $D' \in \tau(D, \alpha(\vec{u}))$ for some ground action $\alpha(\vec{u}) \in \mathit{Act}(U)$, and $D \oplus D' \simeq D'' \oplus D'''$ for some witness ι. We prove that $D''' \in \tau(D'', \alpha(\iota'(\vec{u}))$ where ι' is a bijection extending ι. By the definition of τ in $\mathcal{P}_\Gamma$, $D' \in \tau(D, \alpha(\vec{u}))$ if $(\Sigma_D, e, \Sigma_{D'})$, where e is a ground event with payload $\vec{u}$, and $\alpha_i = \alpha_e$ for one of the components in α. By Lemma 10 we have that $(\Sigma_{D''}, \iota'(e), \Sigma_{D'''})$, and again by definition of τ we obtain that $D''' \in \tau(D'', \alpha(\iota'(\vec{u})))$. As a result, $\mathcal{P}_\Gamma$ is uniform. □

Bibliography

[AHK02] Rajeev Alur, Thomas A. Henzinger, and Orna Kupferman. Alternating-time Temporal Logic. *Journal of the ACM*, 49(5):672–713, 2002.

[AHV95a] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases.* Addison-Wesley, 1995.

[AHV95b] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases.* Addison Wesley Publ. Co., 1995.

[AMPS98] E. Asarin, O. Maler, A. Pnueli, and J. Sifakis. Controller Synthesis for Timed Automata. In *Proc. IFAC Symposium on System Structure and Control*, pages 469–474, 1998.

[AT04] Rajeev Alur and Salvatore La Torre. Deterministic generators and games for LTL fragments. *ACM Trans. Comput. Log.*, 5(1):1–25, 2004.

[AVFY00] Serge Abiteboul, Victor Vianu, Bradley S. Fordham, and Yelena Yesha. Relational Transducers for Electronic Commerce. *J. Comput. Syst. Sci.*, 61(2):236–269, 2000.

[BAPM83] Mordechai Ben-Ari, Amir Pnueli, and Zohar Manna. The Temporal Logic of Branching Time. *Acta Informatica*, 20:207–226, 1983.

[BCG$^+$05] Daniela Berardi, Diego Calvanese, Giuseppe De Giacomo, Richard Hull, and Massimo Mecella. Automatic Composition of Transition-based Semantic Web Services with Messaging. In *VLDB*, pages 613–624, 2005.

[BCJ$^+$97] Anca Browne, Edmund M. Clarke, Somesh Jha, David E. Long, and Wilfredo R. Marrero. An Improved Algorithm for the Evaluation of Fixpoint Expressions. *Theor. Comput. Sci.*, 178(1-2):237–255, 1997.

[BCM$^+$92] Jerry R. Burch, Edmund M. Clarke, Kenneth L. McMillan, David L. Dill, and L. J. Hwang. Symbolic Model Checking: $10^2$0 States and Beyond. *Information and Computation*, 98(2):142–170, 1992.

[BEM97] Ahmed Bouajjani, Javier Esparza, and Oded Maler. Reachability Analysis of Pushdown Automata: Application to Model-Checking. In *CONCUR*, pages 135–150, 1997.

[BG04] Achim Blumensath and Erich Grädel. Finite Presentations of Infinite Structures: Automata and Interpretations. *Theory Comput. Syst.*, 37(6):641–674, 2004.

[BG09] Blai Bonet and Hector Geffner. Solving pomdps: Rtdp-bel vs. point-based algorithms. In *IJCAI*, pages 1641–1646, 2009.

[BHV04] Ahmed Bouajjani, Peter Habermehl, and Tomás Vojnar. Abstract Regular Model Checking. In *CAV*, pages 372–386, 2004.

[BJNT00] Ahmed Bouajjani, Bengt Jonsson, Marcus Nilsson, and Tayssir Touili. Regular Model Checking. In *Proc. of CAV*, pages 403–418, 2000.

[BK08] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking (Representation and Mind Series).* The MIT Press, 2008.

[BL69] J.R. Büchi and L.H. Landweber. Solving sequential conditions by finite-state strategies. *Trans. Amer. Math. Soc.*, 138:295–311, 1969.

[BL09] Francesco Belardinelli and Alessio Lomuscio. Quantified epistemic logics for reasoning about knowledge in multi-agent systems. *Artif. Intell.*, 173(9-10):982–1013, 2009.

[BLP11a] F. Belardinelli, A. Lomuscio, and F. Patrizi. A Computationally-Grounded Semantics for Artifact-Centric Systems and Abstraction Results. In *Proc. of IJCAI*, 2011.

[BLP11b] F. Belardinelli, A. Lomuscio, and F. Patrizi. Verification of Deployed Artifact Systems via Data Abstraction. In *Proc. of ICSOC*, 2011.

[BLP12] F. Belardinelli, A. Lomuscio, and F. Patrizi. An Abstraction Technique for the Verification of Artifact-Centric Systems. In *Proc. of KR*, 2012. To Appear.

[BLS11] Andreas Bauer, Martin Leucker, and Christian Schallhart. Runtime verification for LTL and TLTL. *ACM Transactions on Software Engineering and Methodology*, 20(4):14, 2011.

[Bry86] Randal E. Bryant. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Trans. Computers*, 35(8):677–691, 1986.

[Büc62] J.R. Büchi. On a Decision Method in Restricted Second Order Arithmetic. In *Proc. of Int. Congress for Logic, Methodology and Philosophy of Science*, pages 1–11, 1962.

[BV81] Catriel Beeri and Moshe Y. Vardi. The Implication Problem for Data Dependencies. In *Proc. of ICALP'81*, volume 115 of *LNCS*, pages 73–85. Springer, 1981.

[Cau02] Didier Caucal. On Infinite Terms Having a Decidable Monadic Theory. In *MFCS*, pages 165–176, 2002.

[Cau03] Didier Caucal. On Infinite Transition Graphs having a Decidable Monadic Theory. *Theoretical Computer Science*, 290(1):79–115, 2003.

[CC77] Patrick Cousot and Radhia Cousot. Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. In *POPL*, pages 238–252, 1977.

[CDGL$^+$11] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, Antonella Poggi, Mariano Rodriguez-Muro, Riccardo Rosati, Marco Ruzzi, and Domenico Fabio Savo. The Mastro system for ontology-based data access. *Semantic Web Journal*, 2011. To appear.

[CDLR09] Mika Cohen, Mads Dam, Alessio Lomuscio, and Francesco Russo. Abstraction in model checking multi-agent systems. In *AAMAS (2)*, pages 945–952, 2009.

[CES86] Edmund M. Clarke, E. Allen Emerson, and A. Prasad Sistla. Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986.

[CGJ$^+$03] Edmund M. Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-guided abstraction refinement for symbolic model checking. *J. ACM*, 50(5):752–794, 2003.

[CGL94] Edmund M. Clarke, Orna Grumberg, and David E. Long. Model checking and abstraction. *ACM Trans. Program. Lang. Syst.*, 16(5):1512–1542, 1994.

[CGP99] Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model Checking*. The MIT Press, Cambridge, MA, USA, 1999.

[CGP00] Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model Checking*. The MIT Press, 2000.

[CGRR10] P. Cangialosi, G. De Giacomo, R. De Masellis, and R. Rosati. Conjunctive Artifact-Centric Services. In *ICSOC*, pages 318–333, 2010.

[Chu62] A. Church. Logic, Arithmetics, and Automata. In *Proc. of International Congress of Mathematicians*, 1962.

[Cou94] Bruno Courcelle. Monadic Second-Order Definable Graph Transductions: a Survey. *Theoretical Computer Science*, 126(1):53–75, 1994.

[CW96] Edmund M. Clarke and Jeannette M. Wing. Formal Methods: State of the Art and Future Directions. *ACM Computing Surveys*, 28(4):626–643, 1996.

[Dam94] Mads Dam. CTL* and ECTL* as Fragments of the Modal *mu*-Calculus. *Theoretical Computure Science*, 126(1):77–96, 1994.

[DDV11] Elio Damaggio, Alin Deutsch, and Victor Vianu. Artifact systems with data dependencies and arithmetic. In *ICDT*, pages 66–77, 2011.

[DHPV09] Alin Deutsch, Richard Hull, Fabio Patrizi, and Victor Vianu. Automatic verification of data-centric business processes. In *ICDT*, pages 252–267, 2009.

[DMS$^+$05] Alin Deutsch, Monica Marcus, Liying Sui, Victor Vianu, and Dayou Zhou. A Verifier for Interactive, Data-Driven Web Applications. In *SIGMOD Conference*, pages 539–550, 2005.

[DSV07] Alin Deutsch, Liying Sui, and Victor Vianu. Specification and verification of data-driven Web applications. *J. Comput. Syst. Sci.*, 73(3):442–474, 2007.

[DSVZ06] Alin Deutsch, Liying Sui, Victor Vianu, and Dayou Zhou. A system for specification and verification of interactive, data-driven web applications. In *SIGMOD Conference*, pages 772–774, 2006.

[DV99] Giuseppe De Giacomo and Moshe Y. Vardi. Automata-theoretic approach to planning for temporally extended goals. In *ECP*, pages 226–238, 1999.

[ea11] R. Hull et al. Business Artifacts with Guard-Stage-Milestone Lifecycles: Managing Artifact Interactions with Conditions and Events. In *Proc. of DEBS*, 2011.

[EC80] E. Allen Emerson and Edmund M. Clarke. Characterizing Correctness Properties of Parallel Programs Using Fixpoints. In *Proc. of ICALP*, pages 169–181, 1980.

[EFH$^+$03] Cindy Eisner, Dana Fisman, John Havlicek, Yoad Lustig, Anthony Mcisaac, and David Van Campenhout. Reasoning with temporal logic on truncated paths. pages 27–39. 2003.

[EH86] E. Allen Emerson and Joseph Y. Halpern. "Sometimes" and "Not Never" revisited: on branching versus linear time temporal logic. *Journal of the ACM*, 33(1):151–178, 1986.

[Eme96] E. Allen Emerson. Automated Temporal Reasoning about Reactive Systems. In Faron Moller and Graham Birtwistle, editors, *Logics for Concurrency: Structure versus Automata*, volume 1043 of *LNCS*, pages 41–101. Springer, 1996.

[FdGL12] P. Felli, G. de Giacomo, and A. Lomuscio. Synthesizing Agent Protocols From LTL Specifications against Multiple Partially-Observable Environments . In *Proc. of KR*, 2012. To Appear.

[FHMV95] R. Fagin, J. Y. Halpern, Y. Moses, and M. Y. Vardi. *Reasoning About Knowledge.* The MIT Press, 1995.

[FKMP05] Ronald Fagin, Phokion G. Kolaitis, Renée J. Miller, and Lucian Popa. Data Exchange: Semantics and Query Answering. *Theor. Comp. Sci.*, 336(1):89–124, 2005.

[FL02] Alain Finkel and Jérôme Leroux. How to Compose Presburger-Accelerations: Applications to Broadcast Protocols. In *FSTTCS*, pages 145–156, 2002.

[GFPS10] Giuseppe De Giacomo, Paolo Felli, Fabio Patrizi, and Sebastian Sardiña. Two-Player Game Structures for Generalized Planning and Agent Composition. In *AAAI*, 2010.

[GMP09] Giuseppe De Giacomo, Riccardo De Masellis, and Fabio Patrizi. Composition of Partially Observable Services Exporting their Behaviour. In *ICAPS*, 2009.

[GP09] Giuseppe De Giacomo and Fabio Patrizi. Automated Composition of Nondeterministic Stateful Services. In *WS-FM*, pages 147–160, 2009.

[Gro01] M. Grohe. Generalized model-checking problems for first-order logic. In A. Ferreira and H. Reichel, editors, *STACS*, volume 2010 of *Lecture Notes in Computer Science*, pages 12–26. Springer, 2001.

[GS84] F. Gcseg and M. Steinby. Tree Automata, 1984.

[GS97] Susanne Graf and Hassen Saïdi. Construction of Abstract State Graphs with PVS. In *CAV*, pages 72–83, 1997.

[GvdM04] P. Gammie and R. van der Meyden. MCK: Model checking the logic of knowledge. In *Proceedings of 16th International Conference on Computer Aided Verification (CAV'04)*, volume 3114 of *LNCS*, pages 479–483. Springer-Verlag, 2004.

[HCG+11] B. Bagheri Hariri, D. Calvanese, G. De Giacomo, R. De Masellis, and P. Felli. Foundations of Relational Artifacts Verification. In *Proc. of BPM*, 2011.

[HD11] Yuxiao Hu and Giuseppe De Giacomo. Generalized planning: Synthesizing plans that work for multiple environments. In *IJCAI*, pages 918–923, 2011.

[HR99] Michael R. A. Huth and Mark Ryan. *Logic in Computer Science: Modelling and Reasoning About Systems.* Cambridge University Press, 1999.

[HRS05] A. Harding, M. Ryan, and P.-Y. Schobbens. A new algorithm for strategy synthesis in ltl games. In N. Halbwachs and L. D. Zuck, editors, *TACAS*, volume 3440 of *Lecture Notes in Computer Science*, pages 477–492. Springer, 2005.

[HVCG07] S. Hallé, R. Villemaire, O. Cherkaoui, and B. Ghandour. Model checking data-aware workflow properties with ctl-fo+. In *EDOC*, pages 267–278. IEEE Computer Society, 2007.

[JN00] Bengt Jonsson and Marcus Nilsson. Transitive Closures of Regular Relations for Verifying Infinite-State Systems. In *TACAS*, pages 220–234, 2000.

[JvdH04] Wojciech Jamroga and Wiebe van der Hoek. Agents that know how to play. *Fundam. Inform.*, 63(2-3):185–219, 2004.

[KLP04] Magdalena Kacprzak, Alessio Lomuscio, and Wojciech Penczek. From Bounded to Unbounded Model Checking for Temporal Epistemic Logic. *Fundamenta Informaticae*, 63(2-3):221–240, 2004.

[KM91] Hirofumi Katsuno and Alberto Mendelzon. On the Difference Between Updating a Knowledge Base and Revising It. In *Proc. of KR'91*, pages 387–394, 1991.

[KMM+01] Yonit Kesten, Oded Maler, Monica Marcus, Amir Pnueli, and Elad Shahar. Symbolic model checking with rich assertional languages. *Theor. Comput. Sci.*, 256(1-2):93–112, 2001.

[Kol05] Phokion G. Kolaitis. Schema Mappings, Data Exchange, and Metadata Management. In *Proc. of PODS 2005*, pages 61–75, 2005.

[Koz83] Dexter Kozen. Results on the Propositional mu-Calculus. *Theoretical Computoter Science*, 27:333–354, 1983.

[KV00] Orna Kupferman and Moshe Y. Vardi. Synthesis with incomplete informatio. In *In Advances in Temporal Logic*, pages 109–127. Kluwer Academic Publishers, 2000.

[Len02] Maurizio Lenzerini. Data Integration: A Theoretical Perspective. In *Proc. of PODS 2002*, pages 233–246, 2002.

[LPP70] David C. Luckham, David Michael Ritchie Park, and Mike Paterson. On Formalised Computer Programs. *J. of Computer and System Sciences*, 4(3):220–249, 1970.

[LPQ10] A. Lomuscio, W. Penczek, and H. Qu. Partial Order Reductions for Model Checking Temporal-epistemic Logics over Interleaved Multi-agent Systems. *Fundamenta Informaticae*, 101(1-2):71–90, 2010.

[LQR09a] A. Lomuscio, H. Qu, and F. Raimondi. Mcmas: A model checker for the verification of multi-agent systems. In A. Bouajjani and O. Maler, editors, *CAV*, volume 5643 of *Lecture Notes in Computer Science*, pages 682–688. Springer, 2009.

[LQR09b] Alessio Lomuscio, Hongyang Qu, and Franco Raimondi. MCMAS: A Model Checker for the Verification of Multi-Agent Systems. In *CAV*, pages 682–688, 2009.

[LQS08a] Alessio Lomuscio, Hongyang Qu, and Monika Solanki. Towards Verifying Compliance in Agent-based Web Service Compositions. In *AAMAS (1)*, pages 265–272, 2008.

[LQS08b] Alessio Lomuscio, Hongyang Qu, and Monika Solanki. Towards Verifying Contract Regulated Service Composition. In *ICWS*, pages 254–261, 2008.

[LQSS07] Alessio Lomuscio, Hongyang Qu, Marek J. Sergot, and Monika Solanki. Verifying Temporal and Epistemic Properties of Web Service Compositions. In *ICSOC*, pages 456–461, 2007.

[LR06] Alessio Lomuscio and Franco Raimondi. Model checking knowledge, strategies, and games in multi-agent systems. In *AAMAS*, pages 161–168, 2006.

[MP06] Angelo Montanari and Gabriele Puppis. Verification of Infinite State Systems. Lecture Notes for the 18th Summer School in Logic, Language and Information (ESSLLI'06), 2006.

[MS85] David E. Muller and Paul E. Schupp. The Theory of Ends, Pushdown Automata, and Second-Order Logic. *Theor. Comput. Sci.*, 37:51–75, 1985.

[Pnu81] Amir Pnueli. A Temporal Logic of Concurrent Programs. *Theoretical Computer Science*, 13:45–60, 1981.

[PPS06a] N. Piterman, A. Pnueli, and Y. Sa'ar. Synthesis of reactive(1) designs. In E. Allen Emerson and Kedar S. Namjoshi, editors, *VMCAI*, volume 3855 of *Lecture Notes in Computer Science*, pages 364–380. Springer, 2006.

[PPS06b] Nir Piterman, Amir Pnueli, and Yaniv Sa'ar. Synthesis of Reactive(1) Designs. In *VMCAI*, pages 364–380, 2006.

[PR85] R. Parikh and R. Ramanujam. Distributed processes and the logic of knowledge. In *Logic of Programs*, pages 256–268, 1985.

[PR89a] Amir Pnueli and Roni Rosner. On the Synthesis of a Reactive Module. In *POPL*, pages 179–190, 1989.

[PR89b] Amir Pnueli and Roni Rosner. On the Synthesis of a Reactive Module. In *Proc. of POPL'89*, pages 179–190, 1989.

[PS11] Eric Pacuit and Sunil Simon. Reasoning with protocols under imperfect information. *Review of Symbolic Logic*, 4(3):412–444, 2011.

[PSZ10] Amir Pnueli, Yaniv Sa'ar, and Lenore D. Zuck. JTLV: A Framework for Developing Verification Algorithms. In *CAV*, pages 171–174, 2010.

[Rab72] M.O. Rabin. *Automata on Infinite Objects and Church's Problem.* American Mathematical Society, Boston, MA, USA, 1972.

[Rei01] Raymond Reiter. *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems.* The MIT Press, 2001.

[RFV95] Yoram Moses Ronald Fagin, Joseph Y. Halpern and Moshe Y. Vardi. *Reasoning about Knowledge.* MIT Press, Cambridge, MA, 1995.

[RN10] Stuart J. Russell and Peter Norvig. *Artificial Intelligence - A Modern Approach (3. internat. ed.).* Pearson Education, 2010.

[RV10] Kristin Y. Rozier and Moshe Y. Vardi. LTL Satisfiability Checking. *International Journal on Software Tools for Technology Transfer*, 12(2):123–137, 2010.

[Saï00] Hassen Saïdi. Model Checking Guided Abstraction and Analysis. In *SAS*, pages 377–396, 2000.

[SC85] A. Prasad Sistla and Edmund M. Clarke. The Complexity of Propositional Linear Temporal Logics. *J. ACM*, 32(3):733–749, 1985.

[Sch02] P. Schnoebelen. The complexity of temporal logic model checking. In P. Balbiani, N. Suzuki, F. Wolter, and M. Zakharyaschev, editors, *Advances in Modal Logic*, pages 393–436. King's College Publications, 2002.

[Sem84] Alexei L. Semenov. Decidability of Monadic Theories. In *MFCS*, pages 162–175, 1984.

[Spi03] Marc Spielmann. Verification of relational transducers for electronic commerce. *J. Comput. Syst. Sci.*, 66(1):40–65, 2003.

[Var96] Moshe Vardi. An automata-theoretic approach to linear temporal logic. In Faron Moller and Graham Birtwistle, editors, *Logics for Concurrency*, volume 1043 of *Lecture Notes in Computer Science*, pages 238–266. Springer Berlin / Heidelberg, 1996.

[vdHW02] Wiebe van der Hoek and Michael Wooldridge. Tractable multiagent planning for epistemic goals. In *AAMAS*, pages 1167–1174, 2002.

[vdM96] Ron van der Meyden. Finite state implementations of knowledge-based programs. In Vijay Chandru and V. Vinay, editors, *FSTTCS*, volume 1180 of *Lecture Notes in Computer Science*, pages 262–273. Springer, 1996.

[vdMS99] Ron van der Meyden and Nikolay V. Shilov. Model Checking Knowledge and Time in Systems with Perfect Recall. In *FSTTCS*, pages 432–445, 1999.

[Via09] Victor Vianu. Automatic verification of database-driven systems: a new frontier. In *ICDT*, pages 1–13, 2009.

[Wal00] Igor Walukiewicz. Model Checking CTL Properties of Pushdown Systems. In *FSTTCS*, pages 127–138, 2000.

[Wal02] Igor Walukiewicz. Monadic Second-Order Logic on Tree-like Structures. *Theoretical Computer Science*, 275(1-2):311–346, 2002.

[WL01] M. Wooldridge and A. Lomuscio. A computationally grounded logic of visibility, perception, and knowledge. *Logic Journal of the IGPL*, 9(2):273–288, 2001.

[Woo00] M. Wooldridge. Computationally Grounded Theories of Agency. In *Proc. of ICMAS*, pages 13–22. IEEE Press, 2000.

[Woo09] Michael J. Wooldridge. *An Introduction to MultiAgent Systems (2. ed.)*. Wiley, 2009.