Modeling and verifying choreographed multi-agent-based web service compositions regulated by commitment protocols

Warda El Kholy a, Jamal Bentahar a,⁎, Mohamed El Menshawy a,b, Hongyang Qu c, Rachida Dssouli a

a Faculty of Engineering and Computer Science, Concordia University, Canada
b Faculty of Computers and Information, Menofia University, Egypt
c Department of Automatic Control and Systems Engineering, Sheffield University, UK

A R T I C L E   I N F O

Article history:
Available online 17 June 2014

Keywords:
Multi-agent systems
Conditional commitments
Commitment protocols
Agent-based web services
Symbolic model checking
 Verification

A B S T R A C T

The competency to compose web services from available services is one of the most crucial problems in the paradigm of service-oriented computing. Conventional software engineering approaches and even standard languages compose web services as workflow models that control the business logic required to coordinate data over participating services. Such models would not apply to the design of multi-agent based web services, which offer high-level abstractions that support autonomy, business-level compliance, and flexible dynamic changes. In this article, we model interactions among multi-agent based web services by commitment modalities in the form of contractual obligations and devote multi-agent commitment protocols to regulate such interactions and engineer services composition. We develop and fully implement an automatic verifier by enriching the MCMAS model checker with certain symbolic algorithms to verify the correctness of protocols, given properties expressed in a temporal commitment logic, suitably extended with actions. We analyze the time and space complexity of the verifier. Finally, we present the experimental results of two case studies, adopted to check the verifier’s efficiency and scalability.

© 2014 Elsevier Ltd. All rights reserved.

1. Introduction

Web service composition is the capability to aggregate multiple services into a composite service to actually promote a certain functionality that would not have seemingly been possible by a single service. For example, Orbitz and Expedia aggregate hotel reservation, car rental, and airline booking services. A service composition is not only composing functionalities of a set of certain services, but it should also take into account customers’ needs and goals as constraint and motivation for sustainable construction of the composition process. The new composite services could be utilized as single services in other compositions to satisfy customers’ needs (requests). In this context, interaction is the key foundation of composite services and service-oriented architecture (SOA), which basically provides a conceptual model for understanding and implementing web services and relationships among the components of this model. This is because valuable composite services will emerge from the interaction of more specialized services, and the SOA architecture is indeed based on the interaction among three different components (service provider, service requester, and service registry) to describe, publish, invoke, and register services. In a point of matter, there have been several predominate proposals supporting this claim. For examples: (1) web services forming a certain community must be capable of interacting so as to collaborate and compose with each other to carry out allocated tasks and satisfy common goals (Khoshavifar et al., 2013); and (2) the interaction is the basic aspect to separate the composition functioning of various web services into operational behavior that defines the composition model, and control behaviors that express desirable properties (Bentahar, Yahyaoui, Ková, & Maamar, 2013). The importance of the interaction is also made clear through practical and industrial notions of modeling composite services (Barros, Dumas, & Oaks, 2005): orchestration, and choreography. Orchestration describes how multiple services can interact with each other from the perspective of a single participant using invocation-based approaches. This single participant acts as an orchestrator, coordinating the invoked services and linking the results computed by them. It also uses traditional program- ming constructs (such as loops, conditional branches, forks, and joins) to handle data received/sent from/to partner web services. Furthermore, orchestration refers to automated execution of a workflow in the sense that it produces executable business processes that could interact with internal and external services. Since
orchestration does not describe a coordinated set of interactions between partners, but rather the execution of a specific business process, it only provides a local view. This local view of orchestration is not sufficient to specify interaction protocols; instead a global view of the interaction among participating web services has been acknowledged (Baldoni, Baroglio, Martelli, & Patti, 2007; Chopra & Singh, 2013, chap. 3). This level of specification is captured by the notion of choreography, which particularly imposes a legal sequence of messages exchanged among participants (Yeung, 2011). The business process execution language for web services (WS-BPEL) is an orchestration language. Examples of choreography languages include web service choreography interface (WSCl) (Arkin et al., 2002) and web services choreography description language (WS-CDL) (Barros et al., 2005). Differently from orchestration, the focus of choreography is not on generating executable business processes, but rather on specifying the public contracts that supply the necessary rules of business engagements for making all the interacting participants correctly cooperate and collaborate. Fig. 1 graphically shows the relationship between orchestration and choreography.

1.1. Challenges

Although the technology for developing basic services has attained a certain level of maturity (e.g., FedEx and UPS shipping services are now entrusted in e-commerce transactions), there remain open challenges when it comes to engineering composite services. Those challenges concern:

1. The engagement in modeling complex business interactions that go beyond simple sequences of ordered messages from high-level abstractions.
2. The specification of flexible interaction protocols that characterize the global behaviors of web services composition and regulate the interactions among them in a declarative manner.
3. The automatic detection of undesirable service behaviors (e.g., a web service might subvert or cancel its interaction, malfunction or violate its design functionality), given interaction protocols. In other terms, how can we be certain the systems of composite services will be safe and reliable?

From our perspective, such challenges primarily stem from two technical reasons. The former reason is the growing rate of qualitative business changes in web services so as to keep up-to-date, compete, and achieve the needs of customers. For example, a customer might try to change, update, and cancel an order because of some unexpected conditions or get a refund for a faulty product. In the WS-BPEL language, such kinds of changes are usually specified in different tasks or services, which in turn need compensation tasks to handle their exceptions that appear at the interaction. With the complexity of “domain level logic”, we could not simply augment such tasks or services without considering both the size of the service models and the possibility to separate and distinguish between tasks and compensation ones (Wan & Singh, 2005). The latter reason is the autonomy, with minimal human intervention, of participating services. In other words, services determine for themselves what and when to do interaction with each other. Typically, the degree of autonomy exists because simply web services are deployed in remote and hostile environments, and require to respond much more rapidly than human can. Therefore, designers of web services as a whole system cannot guarantee an ideal outcome of the composite services and they have no overall control over the system.

1.2. Contributions

The contributions of this manuscript are threefold: a formal language and automatic verification tool for commitment protocols, a special type of multi-agent interaction protocols, along with the complexity analysis of the proposed algorithm. Specifically,

1. The language provides the primitives to specify protocols such as roles, messages and their meanings exchanged among roles in terms of commitments and commitment actions. We formally model and compile protocols into guarded automata with shared and unshared variables where each participating service is modeled and compiled into a guarded automaton as well. The formalization enables us to automatically verify protocol properties using model checking.
2. Our tool has an input language and symbolic algorithm. The input language extends the ISPL language of MCMAS (Lomuscio, Qi, & Raimondi, 2009), a symbolic model checker developed to verify MASs, with shared and unshared variables. Our symbolic algorithm extends the standard CTL algorithm (Clarke, Grumberg, & Peled, 1999) implemented in MCMAS with procedures for commitment and fulfillment modalities. Based on capabilities

![Fig. 1. Orchestration vs choreography: orchestration produces an executable process and choreography rules the sequence of messages exchanged between participants (adapted from Peltz, 2003).](image-url)
of MCMAS, our tool provides the validation process before conducting the verification process, which in turn tests whether or not the modeled system of a protocol behaves as intended.

3. A positive result of this algorithm is that its analyzed complexity is P-complete for explicit models and PSPACE-complete for concurrent programs.

The rest of this article is organized as follows. In Section 2, we present the motivations of this work and then discuss related work to identify shortcomings and to support the aforementioned challenges that we address in this manuscript. In Section 3, we briefly present the notion of social commitments, commitment actions, and commitment protocols from our perspective. In Section 4, we present guarded automata of both commitment protocols and participating multi-agent based web services. Such automata extend and improve our extended version (Bentahar, El-Menshawy, Qu, & Dssouli, 2012; El-Kholy, El-Menshawy, Bentahar, Qu, & Dssouli, 2013) of the interpreted system formalism introduced by Fagin, Halpern, Moses, and Vardi (1995) to model MASs. Then, we present the syntax and semantics of the so-called CTL$^{	ext{cc}}$ of social commitments and formal specification of commitment protocols (El-Kholy, El-Menshawy, Bentahar, Qu, & Dssouli, in press). Given that, we develop our symbolic model-checking algorithm (tool) and analyze its computational complexity in Section 5. In Section 6, we present the full implementation of our algorithm on top of the MCMAS tool and then report experimental results of verifying two case studies (the purchase protocol Singh, Chopra, & Desai, 2009 and insurance claim processing Bentahar et al., 2012) in order to test and compare the effectiveness of the developed and implemented algorithm (tool). We discuss, conclude and then identify future work in Section 7.

2. Motivations and related work

2.1. Motivations

Our aim in this manuscript is to propose an approach to address the aforementioned challenges in an integrated framework. The proposed approach specifically acquires the experience of the research community that adopts: (1) the paradigm of multi-agent systems (MASs); and (2) the symbolic model checking techniques. By traversing across these domains, we aim basically to capture the benefits of their cross-fertilization, which in principle demonstrate concerning convergence points. Roughly speaking, MAS strongly provides a mainstream framework for modeling and reasoning about services and their compositions (Lomuscio, Qu, & Solanki, 2012). In fact, the relationship between agents and services is defined by the W3C consortium (Booth et al., 2004) as follows: “a web service is an abstract notion that must be implemented by a concrete agent. The agent is the concrete piece of software that sends and receives messages”. We capture this relationship by calling agent-based web service. Furthermore, MAS theories are appropriate in the consideration of complicated web service interactions and directly support a rich, flexible, and expressive range of specifications. Concretely speaking, we utilize multi-agent social commitments and a special type of multi-agent interaction protocols, namely commitment protocols, which are now widely recognized as a powerful representation and regulation for the interactions among autonomous and heterogeneous agents in MAS (Baldoni, Baroglio, Marengo, & Patti, 2013; Bentahar et al., 2012; El-Menshawy, Bentahar, El-Kholy, & Dssouli, 2013; Singh, 2000). It is worth at this point mentioning that although a choreography supported by WS-CDL (web services choreography description language) (Barros et al., 2005) often specifies and constructs systems that could closely agree with the manner of constructing MASs, it is not suitable for an agent communication theory (Chopra & Singh, 2013, chap. 3). This is because there is no a clear manner to define the meanings of exchanged messages, which are precisely needed to enable an agent to carry out high-level reasoning about protocol actions, and to interoperate effectively with other agents. The interoperability readily means all exchanged messages among agents are understood by each other and their interaction is deadlock-free. Also, WS-CDL adds certain actions into a choreography to guide a participating service, for example, to decide what it ought to do when receiving a message (Chopra & Singh, 2013, chap. 3), but these actions are invariably private. While commitment protocols can play the same role of choreographies, they define the meanings of the interactions in terms of social commitments and social commitment actions. Thus, the engineering of composed systems (MASs) means rigorously and declaratively specifying the interactions among multi-agent based web services by the way of commitment protocols. According to a declarative specification, commitment protocols are flexible, making them suitable for accommodating dynamic changes or exceptions (Chopra & Singh, 2013, chap. 3; Yolum & Singh, 2002), and can be directly verified (El-Menshawy et al., 2013; Venkatraman & Singh, 1999) without requiring the transformation into other formalisms (e.g., Petri nets). Simply put, our approach supports a dynamic service composition using commitment actions, especially assignment and delegation actions, regulated by the same protocol that can be automatically verified against certain composition specifications. Based on these reasons, we associate our models (see Definition 4.4) to commitment protocols to address the limitations of choreographies. On the other hand, orchestrations can be seen as a special kind of our improved choreographies in which all interactions are made between the orchestrating service and one of the partners and no interaction is considered among partners.

In principle, the suggestion of using social commitments into the context of the SOA computing was pioneered by Singh and Huhns in their book (Singh & Huhns, 2005). Informally, commitment rigorously expresses a social engagement made by a debtor agent towards a creditor agent to bring about certain condition (Singh, 2000; Singh & Huhns, 2005) in the kind of contractual obligations (Bentahar et al., 2012; Desai, Narendra, & Singh, 2008; Wan & Singh, 2005). Singh and Huhns’s suggestion has been further investigated by using commitment frameworks and business protocols to extend SOA (Singh et al., 2009), the WS-BPEL language (Wan & Singh, 2005), the web ontology language (OWL) (Desai, Mallya, Chopra, & Singh, 2005), and cross-organizational business processes (Telang & Singh, 2012). However, as we will discuss in details in Section 2.2, those proposals do not integrate formal semantics of commitments and their actions that manipulate commitments. Lacking such formal semantics makes the feasibility of formal verification techniques is very hard or impossible. Also, they do not address the formal verification problem of commitments and business protocols having social semantics. Another technical shortcoming is that current proposals, which utilize the concept of contracts (Lomuscio et al., 2012) and protocols (Fu, Bultan, & Su, 2004) to regulate the composition of multi-agent based web services do not explicitly model and account for interactions among participating services. Consequently.

1. The first aim of this paper is to use our branching-time temporal logic, called CTL$^{	ext{cc}}$, which extends computation tree logic (CTL) with temporal modalities for commitments and their fulfillments (El-Kholy et al., 2013). To complete the picture, Section 2.2 discusses the reasons behind selecting this logic. In this paper, we suitably extend CTL$^{	ext{cc}}$ to deal with other commitment actions, such as cancel, delegate, and assign so as to flexibly
capture more business scenarios. By doing so, the semantics of commitments and all associated actions are formally defined, and contractual commitments explicitly model the interactions among parties in a direct and natural way.

2. The second aim of this paper is to develop a symbolic algorithm to perform model checking CTL\(^*\), which is missing in our previous paper (El-Kholy et al., 2013).

Symbolic model checking (Clarke et al., 1999) is one of the most promising techniques, which in turn help designers of multi-agent based web services detect and then eliminate or repair undesirable service behaviors. It totally proves or disproves that a composition of multi-agent based web services regulated by commitment protocols will satisfy a certain set of preset protocol properties, expressed by our logic. Given a formal model \(M\) for composite multi-agent based web services and a desirable property \(\phi\), model checking symbolically carries out an exhaustive exploration of all potential behaviors of the model to answer the decision question: Does the model \(M\) satisfy the formula \(\phi\)? (formally \(M \models \phi\))? With respect to other verification approaches (e.g., testing), model checking gives designers prefix confidence when desired properties are verified. Further, when a certain property is not satisfied, model checking produces an error trace, termed also a counterexample, which in turn reveals step-by-step how the property is violated by the model. This counterexample is a big advantage of model checking as it provides invaluable information to designers for understanding and in turn repairing or removing undesirable behaviors.

### 2.2. Related work

We group current approaches to model web service composition into conventional software engineering approaches, commitment-based approaches, and multi-agent approaches. The former group strongly advocates traditional techniques, such as AI planning, process algebra, classical logics, and Petri nets. The idea of composing services employed in this group is intimately related to workflow models that control logic required to coordinate data over partner services using different constructs, such as sequence and parallel. The workflow model in the classical planning problem is generally specified as the task of coming up with a sequence of services that achieves a specified goal (Fan, 2013; Kontopoulos, Vrakas, Kokkoras, Bassiliades, & Vlahavas, 2008).

The workflow technique therefore becomes the core element in the leading service composition languages such as WS-BPEL (WS-BPEL), and WS-CDL (Barros et al., 2005). For example, WS-BPEL describes business processes as executable and abstract processes. These processes model and implement the behavior of participating services in the form of a private workflow that controls the flow of data through synchronous and asynchronous invocation methods. They are made abstract to hide their internal behavior or part of it. In fact, usually the abstract process definition is published while the executable implementation is being built. Such traditional techniques especially exploited in WS-BPEL and WS-CDL have been recently criticized in the commitment-based approaches (the second group in our classification). This is because traditional techniques mostly gear for low-level details (i.e., how services exchange messages) to achieve interoperability, rather than offering high-level abstractions (i.e., what the messages mean in the real world).

Among commitment-based approaches, the authors in Singh et al. (2009) improved current SOA architectures, interpreting services narrowly as computational objects to use invocation methods, by giving primacy to the business meanings of service engagements. Computationally, agents perform service engagements (e.g., payment service) by creating and manipulating commitments to one another using a set of commitment actions. In Wan and Singh (2005), web service descriptions are enhanced with commitments and their actions in order to make the continuous updating of service models employed in dynamic and open environments feasible. The authors only concentrated on the formulation of commitments and their actions in the XML document, which in turn leaves the formulation of the message syntax and semantics open. Xing and Singh (2001) proposed a set of commitment patterns inspired by object-design patterns to model agent interactions. Each pattern captures a business scenario, which can be specialized as commitment actions. Commitments are simply modeled as abstract data types where the commitment content is defined as a predicate with a vector of domain arguments to pass data values. On the other hand, commitment actions are modeled as predicates with 3 arguments. The authors then exploit a state-chart to specify the behavior model of each agent and to coordinate generally the interactions of agents. Xing and Singh continued their approach in Xing and Singh (2003) by developing: (1) an algorithm to transform the state-chart of an agent’s behavior model into CTL model; (2) a library of commitment patterns; and (3) a library of agents’ behavior models along with a theorem, proving which behavior models comply with which patterns. Notice that their composition process is based on using the concept of object designs, called “Merge” to compose “multiple state-charts into a state-chart” in order to establish agents’ behavior model (Xing & Singh, 2003). However, the concepts of conditional commitments (i.e., commitments that become active provided some condition holds), and commitment protocols are not considered. From our perspective, the main limitation of the approaches (Singh et al., 2009; Wan & Singh, 2005; Xing & Singh, 2001, 2003) is the lack of automatically checking the compliance of committing agent behaviors with specifications. The proof construction in theorem proving is generally difficult, even if the underlying logic is simple, and requires a good deal of human ingenuity (Emerson, 1990, chap. 16). Also, manual proof construction does not scale up well to large systems. Telang and Singh (2012) tried to solve the problem of verifying agent interactions in the context of cross-organizational business processes (a type of service engagements). They specifically exploited the NuSMV symbolic model checker to check whether or not an operational model, aggregating from a set of business processes interactions, correctly supports business patterns formalized as CTL formulae. However, the way of formally composing the interactions among business processes and the internal design of agents reflecting their local details and implementing business processes are omitted.

**Remark 2.1.** While modeling commitments as abstract data types that can be treated by propositions in CTL model (Xing & Singh, 2001, 2003), and as objects with different states that can be encoded as SMV modules (Telang & Singh, 2012) is easy to implement and treat, it suffers in our perspective from two technical shortcomings: (1) it does not reflect the real and concrete meanings of commitments; and (2) no formal semantics is given to commitments that can be, for example, model checked.

Multi-agent approaches (the third group in our classification) focus on regulating the business interactions among multi-agent based web services within the composition process using contracts, conversation protocols, and business protocols. The concept of contracts captures complex human activities that need sophisticated declarative specifications in order to certify their legal agreements. Contracts can be broken (violated) by one of the participants, which in turn requires applying legal remedies in the form of penalties or repairs. Among multi-agent approaches, (Lomuscio et al., 2012) modeled all possible behaviors of multi-agent based services as well as the correct behaviors for every agent regulated by binding electronic contracts as WS-BPEL
specifications. Such specifications are transformed into ISPL (the input language of MCMAS model checker Lomuscio et al., 2009) to automatically verify desirable properties needed for service compositions and formalized by temporal-epistemic logic. This logic in fact extends CTL with the knowledge modality and local atomic propositions. Technically, local propositions are introduced to address serious difficulties that arise when combining the correct behavior operator with the temporal modalities and knowledge modality. However, the direct interaction between the contract’s parties are not captured and the formalization of the contract itself is abstracted away. Fu et al. (2004) modeled services as agent peers interacting via asynchronous messages in XML format. They then presented a compiler translating the conversation protocol to PROMELA and showed that properties of conversation protocols, expressed in linear temporal logic (LTL), can be automatically verified using the SPIN model checker. However, the automata-based model checker SPIN suffers from the state explosion problem preventing it from verifying complex and large systems. Further, the proposed model concentrates on control flows and data manipulation semantics, which in turn restricts the flexibility and modularity of protocols. Desai et al. (2005) developed a language, called OWL-P, to specify business protocols having social semantics in terms of using commitments. It particularly incorporates the web ontology language (OWL) and a set of primitives to specify protocols such as roles, messages and their meanings exchanged between them. Notice that each business process reflects a composition of a set of business protocols with respect to some properties expressed in pi-calculus. However, OWL-P waives formal semantics for social commitments. Also, they do not consider how to verify composite protocols.

Because our approach combines the main benefits of three research areas (agent communication, symbolic model checking, and web service composition), it is natural to investigate recent approaches that formally define a well-formed semantics for social commitments and associated actions from agent communication perspective. This specifically means that we only consider approaches that model commitments as temporal modalities to be able to represent and reason about them in a declarative manner. Among them, Singh (2008) introduced a model-theoretic semantics for social commitments by extending LTL with temporal modalities for practical conditional commitments that are about what is to be made, and dialectical conditional commitments that are about what holds. In this model, the semantics is interpreted using Segerberg’s idea, which in turn maps “each world into a set of set of worlds”. For example, the semantics of dialectical commitments is delineated by computing the set of moments where the commitment content holds and then testing if those moments are among the moments satisfying the commitment condition. However, model checking this semantics is still an open problem. Verifying Singh’s semantics using model checking needs either defining an equivalent semantics using Kripke structures or defining a completely new model checking approach from scratch, technically because it is not based on standard Kripke structures. We extended CTL with modalities to represent and reason upon unconditional commitments and their fulfillments (Bentahar et al., 2012; El-Menshawy, Bentahar, El-Kholy, & Dssouli, 2013). The resulting logic is called CTLc. We then proceeded to verify the correctness of CTLc using a direct verification technique that develops symbolic algorithms needed for the new modalities (Bentahar et al., 2012) or using a reduction technique that transforms the problem of model checking CTLc into the problem of model checking ACTL (an extension of CTL with action formulae), and the problem of model checking GCTLc (a generalized version of CTLc with action formulae) so that the extended NuSMV symbolic model checker and the CWB-NC automata-based model checker as a benchmark are usable. However, the semantics of other commitment actions, and conditional commitments are not addressed. Through the manuscript we will compare our previous approaches (Bentahar et al., 2012; El-Menshawy et al., 2013) with the present one in a formal way. We also El-Menshawy et al. (2013) developed a logical model based on a new temporal logic, named ACTLc, which in principle extends CTLc with temporal modalities for conditional commitments and associated actions. The reduction technique that formally transforms the problem of model checking ACTLc into the problem of model checking GCTLc so as to use the CWB-NC model checker is also introduced. However, as we showed in Bentahar et al. (2012) and El-Menshawy et al. (2013), formal reduction techniques show very limited scalability and high memory usage. Also, the conditional commitment semantics (El-Menshawy et al., 2013) has a limitation resulting from using the material implication operator to define the casual relationship between conditional and unconditional commitments. This limitation means that a conditional commitment could become active without satisfying its condition. To this end, we select our temporal-commitment logic, called CTLc (an extended version of CTL with conditional commitment and fulfillment modalities) and published as a short paper in El-Kholy et al. (2013), to be our formal language to express commitment protocol properties for the following reasons:

1. The computationally grounded semantics of conditional commitment modality is defined using the standard accessibility relation, which in principle enables us to directly develop model checking algorithm.
2. There are many open model checkers that support CTL model checking algorithm.
3. Although the expressiveness of LTL and CTL are incomparable, the standard model checking algorithms for LTL and CTLc are exponential in the length of the formula and linear in the size of the model (Clarke et al., 1999; Schnoebelen, 2002) and for CTL it is linear in both the size of the formula and model (Clarke et al., 1999; Schnoebelen, 2002). Thus, CTLc balances between the complexity and verification efficiency.
5. The semantics of fulfillment modality solves the paradox appeared in our previous approaches (Bentahar et al., 2012; El-Menshawy et al., 2013) and resulted from the assumption saying that the commitment should be active at the moment of its fulfillment. Indeed, this assumption is not commonly accepted in the literature (see for examples (Singh, 2008; Winikoff, Liu, & Harland, 2005; Yolum & Singh, 2004)).

3. Commitments and commitment protocols

3.1. Commitments

As we outlined in the foregoing, business interactions are typically defined by contracts that describe the roles and responsibilities of their parties along with a list of failure conditions and associated penalties. Most industrial use cases involving contracts are currently analyzed by considering the contract compliant scenarios, described by transition systems (Lomuscio et al., 2012) without investigating the correctness of contracts themselves that are often ambiguous and error prone (Desai et al., 2008). Following Desai et al. (2008), we have two types of contracts: simple and complex. A simple contract can be viewed as a social commitment among its parties, while a collection of simple contracts (or social commitments) forms complex contract. Thus, we understand the interactions via exchanging messages that would occur within the execution of a simple business contract in terms of how they
affect the states of commitment. In general, commitments can be thought of directed obligations that have been manipulated and possibly conditionalized. Differently from modeling commitments as fluents in Desai et al.’s framework, we model commitments as temporal modal operators, which in principle enable us to develop dedicated formal verification techniques that prove safety and reliability for agents to enter into a specific contract.

Some proposals (Desai et al., 2005; Singh, 2000; Wan & Singh, 2005) distinguish between unconditional and conditional commitments. Unconditional commitment \( C'(\text{Debtor, Creditor, Consequent}) \) denotes that the agent-based web service, called debtor, commits to the agent-based web service, called creditor, to bring about the consequent. In other terms, the notation \( C(\text{Debtor, Creditor, Consequent}) \) could mean the debtor service has a contractual commitment to convey the consequent information to the creditor service. The debtor is totally free to decide which path to run to satisfy his consequent. The social aspect that engages the two agents is the key aspect that characterizes a commitment. The basic idea is that the debtor service will only commit towards the creditor service to bring about the consequent if the antecedent holds. The commitment actions provide a principled way to evolve the changes in the social commitment states. Consider the contractual commitment \( C(\text{Mer, Cus, pay 2008, shipGoods}) \) to illustrate the intended meaning of those actions and to define the life cycle of commitment (also called state machine), i.e., the states in which a contract can be in and the possible transitions. A contract can be in one state at a time and it persists in that state until an action is performed on it.

- If the merchant \( \text{Mer} \) ships the goods, i.e., if \( \text{shipGoods} \) holds, the commitment is fulfilled \((\text{Fulfill (Mer, CC)})\) and it is no longer active. So, the state of the merchant contract is changed from the active state into the fulfillment state. Conversely, when there is no way to fulfill a contract, then its active state will be moved to the violation state and the violator services can be penalized.
- Cancel \((\text{Mer, CC})\) is performed by \( \text{Mer} \) to revoke his contract, which becomes no longer active and its active state will be moved to the cancel state.
- Release \((\text{Cus, CC})\) is performed by \( \text{Cus} \) to free \( \text{Mer} \) from his contract, which is no longer active and its active state will be moved to the release state.
- Delegate \((\text{Mer, Mer}_1, \text{CC})\) is performed by \( \text{Mer} \) to shift his role to another merchant \( \text{Mer}_1 \), which creates a new contract \( \text{CC}(\text{Mer}_1, \text{Cus, pay2008, shipGoods}) \) to satisfy the current contract on behalf of \( \text{Mer} \). So, the active state is moved to the delegation state.
- Assign \((\text{Cus, Cus}_1, \text{CC})\) is performed by \( \text{Cus} \) to transfer the contract to another customer \( \text{Cus}_1 \), which becomes the creditor of a new contract. So, the active state is moved to the assignment state.
- If \( \text{Cus} \) makes the payment, i.e., if \( \text{pay2008} \) holds, the contract is transformed \((\text{detached})\) into the unconditional contract \( \text{C(Mer, Cus, shipGoods)} \). And the conditional state \((\text{or active state})\) is moved to the unconditional state.

More recent research work (Singh, 2008; Chopra & Singh, 2013, chap. 3; El-Kholy et al., 2013) considered only conditional commitments, treating unconditional commitments as a special case, where the antecedent is always true. Unlike traditional services that terminate their interactions when a result is delivered (Wan & Singh, 2005) (short-lived interactions), in our approach, it is just the beginning of the life cycle of contracts, created by sending service requests and delivering services, thanks to commitment actions, which in principle allow interacting agents to flexibly modify the underlying service requests and responses (long-lived interactions).

### 3.2. Commitment protocols

While protocols are published publicly, we specify them declaratively in terms of: (1) a set of roles played by autonomous and heterogeneous agents; and (2) a set of messages (also called protocol actions) and their meanings in terms of commitments and associated actions as well as a set of literals that represent the application domain vocabulary of the protocol and can be seen as the set of atomic propositions. By playing a role in the protocol, agents agree about the meanings of messages, terms, and conditions of commitments. Also, we assume that exchanging messages among agents is reliable, which operationally means that messages do not get lost and their orders are preserving. Further, participating agents are still free to perform actions, which are not included in the protocol specification, but their executions cannot: (1) affect the social states holding commitments; and (2) be interpreted as part of the interaction. To clarify the above specification, consider the running case study of the purchase scenario (Singh et al., 2009). Purchasing (say, book) is a business service that combines three individual services: ordering, paying and shipping. We enhanced this scenario with the bank agent to pay on behalf of the customer.

### protocol purchase

role Cus, Mer, Bank

\[\begin{align*}
\text{CTL}^c \text{ Formula pPrice, delBook, sReceipt message}

\text{Cus} \rightarrow \text{Mer:orderMsg means [CC} \\
\text{(Cus,Mer,delBook,pPrice)}]\}

\text{Mer} \rightarrow \text{Cus:deliverMsg means [CC (Cus,Mer,T,pPrice)]}

\text{Cus} \rightarrow \text{Bank:delegateMsg means [CC} \\
\text{(Bank,Mer,T,pPrice)}]

\text{Bank} \rightarrow \text{Mer:payMsg means [Pu (CC} \\
\text{(Bank,Mer,T,pPrice))]}\}

\text{Mer} \rightarrow \text{Cus:receiptMsg means [Inform} \\
\text{(Mer,Cus,sReceipt)]}\}
\end{align*}\]

\footnote{The syntactical rules of CTL\textsuperscript{c} formulae are introduced in Definition 4.5.}

At the outset, we give a name to the protocol \( \text{purchase} \) and then list the names of roles (customer, merchant, etc.) participating in the protocol, and CTL\textsuperscript{c} formulae \( \{\text{pPrice, delBook, etc.}\} \) representing the antecedents and consequences of commitments. Such contracts define the meaning of messages \((\text{orderMsg, deliverMsg, payMsg, etc.})\) sent from a sender to a receiver. Upon sending an order, the customer becomes conditionally contracted to the merchant to pay for the book if it is delivered. The delivery of the book unconditionally commits the customer to pay for it. For some reasons, the customer delegates his contract to the bank to pay on his behalf. When the bank pays, the contract to pay is fulfilled. Finally, the merchant informs the customer by sending the receipt. Observably, commitments reflect business requirements of parties without relinquishing their autonomy in a natural manner: an agent-based web service is expected only to achieve his contract. When the bank agent agrees to pay on behalf of his customer, then he is no more than committed to send the payment to the merchant. The typical use of commitments in protocols involves introducing the syntax and formalization for messages. Commitment protocols, on the other hand, supports flexibility as long as the behavior is correct at the business interaction level. The customer can select to pay first. In this case, one can
add a refund scenario (as a repairable solution) to ensure that the customer has a possibility to receive his payment whenever the merchant is never willing to deliver goods. The modified protocol would continue to function well whenever by the end of the protocol all created commitments are resolved, as discussed in El-Menshawy et al. (2013) and Yolum and Singh (2004). Interestingly, protocols provide clear conceptual boundaries to compose engagements among services. Sending the delegateMsg message results in the termination of the engagement between the customer and merchant and at the same time the establishment of the engagement between bank and merchant. This means the composition is established at the protocol level. Thus, commitment protocols somehow constrain the interactions among participating services in order to make the effective composition. With commitment protocols, the autonomy can be understood as services are free to interact with each other, but they should act rationally to maximize their possessed design objectives while preserving in their perspectives the goals of the composition.

4. Temporal commitment interpreted systems

The formalism of interpreted systems was introduced by Fagin et al. (1995) to provide a mainstream framework for modeling and reasoning about fundamental classes of MASs, such as synchronous and asynchronous. We extended Fagin et al.’s formalism by a set of shared and unshared variables to account for agent communication and to interpret unconditional commitment properties (Bentahar et al., 2012; El-Menshawy et al., 2013). Hereafter, we extend our version of interpreted systems (Bentahar et al., 2012; El-Menshawy et al., 2013) with: (1) an assignment function \( \varphi \); (2) domain values \( D \); and (3) guarded local and global transition functions. The assignment function and domain values are needed to assign values to shared and unshared variables. Technically, these variables are part of agents’ local states and their values along with agents’ actions are used as conditions to control the firing of local and global transitions. As in the original version of the interpreted systems (Fagin et al., 1995), the communication is explicitly implemented by message-passing systems. In our new version of the interpreted system formalism, the MAS is composed of a set \( \mathcal{A} \) of \( m \) agents and an environment \( e \). Each agent \( a \in \mathcal{A} \) implementing a web service supplying certain functionalities is modeled by \( M_a \) as a guarded automaton with variables as follows: \( M_a = (L_a, D_a, \text{Var}_a, \text{Act}_a, P_a, t_a, \mathcal{F}_a, \tau_a) \) such that:

- \( L_a \) is a finite set of states where at any given time each agent-based web service in the system is in a particular local state.
- \( D_a \subseteq \mathbb{N} \) is a domain of natural values.
- \( \text{Var}_a \) is a set of at most \( m - 1 \) local variables (i.e., \( |\text{Var}_a| \leq m - 1 \)) to represent communication channels through which values are sent and received. Each variable from \( \text{Var}_a \) has one value from a domain \( D_a \), but two different variables could have the same value.
- \( \text{Act}_a \) is a set of local actions available to the agent including the commitment actions and null action to account for the temporal evolution of the system.
- \( P_a : L_a \times 2^{\text{Act}_a} \) is a local protocol function producing the set of enabled actions that might be performed by \( a \) in a given local state. It also enables the agent-based web service to consider preferable policies at his local states.
- \( t_a \subseteq L_a \) is the set of initial states of the agent-based web service \( a \).
- \( \mathcal{F}_a \) is a finite set of assignments of the form \( \varphi_j := \varphi_j \), where \( \varphi_j \in \text{Var}_a \), \( t_j \in D_a \), and \( j \in \mathbb{N} \). \( \mathbb{N} \) is the set of positive natural numbers.
- \( \tau_a : L_a \times 2^{\text{Act}_a} \times \text{Act}_a \times \cdots \times \text{Act}_m \times \mathcal{F}_a \rightarrow L_a \) is a guarded local transition function defining a local state from another local state, subset of assignments of the variables, and a tuple \( a = (a_1, \ldots, a_n, a_0) \) termed \textit{joint action} (one for each agent-based web service and environment). The guarded transitions between local states mean that the conditions defined over variables and actions performed by all services in the system must be evaluated to true.

In a similar way, the model \( M_e = (L_e, \text{Var}_e, D_e, \text{Act}_e, P_e, t_0, \mathcal{F}_e, \tau_e) \) is a tuple for an environment \( e \) where \( L_e, \text{Var}_e, D_e, \text{Act}_e, P_e, t_0, \mathcal{F}_e, \) and \( \tau_e \) have the above meanings. It is a special agent that captures any information that might not pertain to a specific agent. We define the semantic of an agent-based web service automaton in terms of its executions as follows. An execution of an agent-based web service \( i \) is an infinite sequence \( (t_i, \text{val}_i^0), (t_i, \text{val}_i^1), (t_i, \text{val}_i^2), \ldots \) where:

1. \( t_i^0 \) is an initial local state of \( i \).
2. \( t_i^j \) is a local state from \( L_i \) and \( \text{val}_i : \text{Var}_i \rightarrow D_i \) is an assignment function for the variables in \( \text{Var}_i \) for \( j \in \mathbb{N} \).
3. For each pairs \( (t_i, \text{val}_i^j) \), there exists a guard \( g \), i.e., a tuple \( (g_{ass}, a) \) where \( g_{ass} \) refers to the set of assignments in \( g \) \( (g_{ass} \subseteq \mathcal{F}_i) \) and \( a \) is a joint action, such that: (1) \( t_i \) rightarrow \( t_i^{j+1} \) is a transition from \( t_i \); and (2) for each variable \( x \in \text{Var}_i \), \( \text{val}_i^{j+1}(x) = v \) if there is an assignment \( (x := v) \in g_{ass} \); otherwise, \( \text{val}_i^{j+1}(x) = \text{val}_i(x) \).

The notion of social state, termed global state in Fagin et al. (1995), represents the instantaneous configuration of all agents in the system at a given time.

**Definition 4.1.** A social state \( s \in S \) is a tuple \( s = (l_1, \ldots, l_m, l_e) \) where each element \( l_e \in L_e \) represents the local state an agent \( i \) is in along with the environment state. The set of all social states \( S \subseteq L_1 \times \cdots \times L_m \times L_e \) is a subset of the Cartesian product of all local states for all agents and environment.

All local transitions are combined together to define a social transition function \( \tau : S \times 2^{\mathcal{F}_1} \times \cdots \times 2^{\mathcal{F}_m} \times \mathcal{F}_e \rightarrow S \) so as to give the overall transition function for the system. One can define \( \tau \) as follows: \( \tau = \tau_1 \times \cdots \times \tau_m \times \tau_e \).

**Definition 4.2.** Suppose \( l_i(s) \) represents the local state of agent \( i \) in the social state \( s \) and the value of a variable \( x \) in the set \( \text{Var}_i \) at \( l_i(s) \) is denoted by \( \text{val}_i^j(s) \). If \( l_i(s) = l_i(s') \) then \( \text{val}_i^j(s) = \text{val}_i^j(s') \) for all \( x \in \text{Var}_i \).

For two agents \( i \) and \( j \) to communicate, they should share a communication channel, which is represented by a shared variable between them.

**Definition 4.3.** A synchronous communication channel between \( i \) and \( j \) does exist iff \( \text{Var}_i \cap \text{Var}_j \neq \emptyset \).

For the variable \( x \in \text{Var}_i \cap \text{Var}_j \), \( l_i(s) = l_j(s') \) means the values of \( x \) in \( l_i(s) \) for \( i \) and in \( l_j(s') \) for \( j \) are the same. Our temporal model composed by the Cartesian product of \( m \) service automata \( M_1, \ldots, M_m \) synchronizes by joint actions.

**Definition 4.4 (Models).** A model of commitment protocols, regulating the composition of multi-agent based web services, \( M = (S, I, T, \{\pi_{ij}| (i, j) \in \mathcal{G}(P)\}, V) \) is a tuple where:
− \( S \subseteq L_1 \times \cdots \times L_m \times L_s \) is a set of reachable social states for the system. Intuitively, it stores the complete information about the influences of sending and receiving messages exchanged by services.

− \( I \subseteq L_1 \times \cdots \times L_m \times L_s \) is a set of initial social states for the system such that \( I \subseteq S \).

− \( T \subseteq S \times S \) is a total temporal relation defined by \( (s, s') \in T \) if \( \tau(s, a_s, \ldots, a_m, a_s, a_s, a_s) = s' \) for some assignments \( a_s \in 2^{I_i} \) and joint action \( (a_1, \ldots, a_m) \).

− For each pair \((i, j) \in A_g^2\), \( _i \mathbb{R}_j \subseteq S \times S \) is a serial social accessibility relation defined by \( s_{i,j} \subseteq S \times S \) if

1. \( l_i(s) = l_i(s') \);
2. \( (s, s') \in T \);
3. \( Var_i \cap Var_j \neq \emptyset \) and \( \forall x \in Var_i \cap Var_j \) we have \( l_i'(s) = l_i'(s') \); and
4. \( \nu : PV \to 2^I \) is a function defining which atomic propositions hold from the set \( PV \) in protocol states.

In Bentahar et al. (2012) and El-Menshawy et al. (2013), we did not consider the reachability condition (condition 2). So, the new definition of accessibility relation is different. In fact, this condition contributes to solve the fulfillment paradox, appeared in Bentahar et al. (2012), El-Menshawy et al. (2013) and discussed in Section 2.2. The intuition of \( s_{i,j} \subseteq S \) is that for a commitment to take place, a communication channel should exist between \( i \) and \( j \) through the shared variable and the accessible state \( s' \) is reachable using \( T \) and indistinguishable from the current state \( s \) for all \( i \) as \( i \) is the agent who is committing; however, for \( j \) who is receiving the commitment, the two states are different as new information is obtained from \( i \) through the communication channel and this is why in the accessible state, \( j \) has the same value as \( i \) has for the shared variable. Furthermore, the accessible state is not completely different from the current state for \( j \) as some information are still the same, and this is why for the unshared variables, the current and accessible states for \( j \) are indistinguishable.

Fig. 2 depicts an example of setting up a communication channel among two social states \((s, s')\). In this example, the merchant and customer are communicating and local states, local actions and the shared and unshared variables for them are as follows. Merchant \((Mer)\): \( l_{mer} = \{ l_mer \} \), \( Act_{mer} = \{ Cancel \} \), and \( Var_{mer} = \{ x \} \). Customer \((Cus)\): \( l_{cus} = \{ l_{cus} \} \), \( Act_{cus} = \{ null \} \), and \( Var_{cus} = \{ x, y \} \). Since there exists a shared variable \( x \) between \( Mer \) and \( Cus \), then we use it to represent and establish the communication channel between such agents, \( y \) is a \( Cus ' s \) unshared variable with \( Mer \). Furthermore, the value of \( x \) for \( Cus \) in \( s' \) is changed to be equal to the value of this variable for \( Mer \) \( (l'_{cus}(s') = l_{mer}(s) ≠ 0) \). However, the value of \( y \) for \( Cus \) at \( s' \) is unchanged. It is worth mentioning that \( s' \) is accessible from \( s \) (i.e., \( s_{mer, cus} \subseteq s \)), because the four conditions of the accessibility relation are meet: (1) \( l_{mer}(s) = l_{mer}(s') \), which is \( l_{mer} \); (2) \( (s, s') \in T \); (3) \( l'_{mer}(s) = l'_{mer}(s') = 0 \); and (4) \( l'_{mer}(s) = l'_{mer}(s') = 1 \).

A composition of a finite set of multi-agent based web services interacting with each other is regulated by the commitment protocol models. Each social state (or protocol state) captures an observable social behavior of composing services and the set of all observable behaviors forms the set of social states. A well-formed infinite sequence \( \tau = s_0, s_1, \ldots \) of social states in \( S \) such that \( \forall \tau \in \tau \), \( s_i \in T \) is termed a path. Thus, the protocol model \( M \) can semantically unwind into a set of computation paths that it allows.

**Definition 4.5 (Syntax).** The syntax of CTL^c^ is given by the following BNF grammar:

\[
\phi ::= p | \neg \phi | \phi \lor \psi | \exists X \phi | \exists G \phi | \exists E \psi | \exists F \psi | \exists C(i, j, \psi, \phi) | \exists F(U(i, j, \psi, \phi))
\]

where \( p \in PV \) is an atomic proposition; \( E \) is the existential quantifier on paths; \( X, G \) and \( U \) are CTL path modal connectives standing for “next”, “globally”, and “until”; and \( \neg, \lor, CC \) and \( Fu \) stand for Boolean connectives, commitments and their fulfillments, respectively.

In this logic, \( CC(i, j, \psi, \phi) \) is read as “agent \( i \) commits towards agent \( j \) that \( \phi \) when the antecedent \( \psi \) holds.” The antecedent \( \psi \) and the consequent \( \phi \) in the context of commitment modality can be any arbitrary formula; so they would be conditional commitments as well. The commitment modalities explicitly model contracts and interactions among their parties, which are missing in Lomuscio et al. (2012). The modality \( Fu(CC(i, j, \psi, \phi)) \) is read as “the commitment \( CC(i, j, \psi, \phi) \) is fulfilled”. Other Boolean connectives and temporal modalities can be defined in terms of the above as usual. For example, \( \exists T \equiv (p \lor \neg p) \), \( \phi \ominus \psi \equiv \neg \phi \lor \psi \), \( EE \phi \equiv E(\exists T U \phi) \), \( AX \phi \equiv \neg EX \phi \), and \( AG \phi \equiv \neg EF \phi \) where \( \ominus \), \( \lor \), and \( \exists T \) stand for implication, and eventually operators respectively.

**Example 4.1.** The following formula representing the simple business contract between an insurance company and patient:

\[
AG(CC(Ins. \ Pat. \ A(-Reimburse U(\text{Approve} \wedge \neg \text{Reimburse}})).
\]

EXEF Reimburse

which states that the insurance company commits to reimburse a covered patient for a health procedure only if the patient obtains an approval from the company prior to the health procedure.
Table 1
Formal specification of commitment protocols using BNF.

<table>
<thead>
<tr>
<th>Protocol</th>
<th>::=</th>
<th>protocol</th>
<th>ProName</th>
<th>RoleNames</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>::=</td>
<td>CTL^c, atomic proposition</td>
</tr>
<tr>
<td>ProName</td>
<td>::=</td>
<td>CTL^c, atomic proposition</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RoleNames</td>
<td>::=</td>
<td>Role</td>
<td>Role</td>
<td>Role</td>
</tr>
<tr>
<td>FormulaNames</td>
<td>::=</td>
<td>Formula</td>
<td>Formula</td>
<td></td>
</tr>
<tr>
<td>MessageNames</td>
<td>::=</td>
<td>Message</td>
<td>Message</td>
<td></td>
</tr>
<tr>
<td>Message</td>
<td>::=</td>
<td>Sender -&gt; Receiver: MsgName means{Meaning}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sender</td>
<td>::=</td>
<td>Role</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Receiver</td>
<td>::=</td>
<td>Role</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Meaning</td>
<td>::=</td>
<td>CC(DBtr. Cdr. Ant. Csq), CC(DBtr. Cdr. Ant. Csq)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dbtr</td>
<td>::=</td>
<td>Role</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cdr</td>
<td>::=</td>
<td>Role</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ant</td>
<td>::=</td>
<td>Formula</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Csq</td>
<td>::=</td>
<td>Formula</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CTL^c atomic propositions</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Example 4.2. The simple business contract $AG(CC(Wrh. Str. bill. EOGoods))$ means that a warehouse commits to supply ordered goods to a retail store only after receiving a bill for the order amount.

Example 4.3. The complex business contract $AG(CC(Trv. All. CC (All. Trv. pAgree. Confirm) \land pPay. tpay))$ states that a travel agent commits to send the payment if: (1) the airline commits to confirm the trip after receiving the agreement from the passenger; and (2) the passenger sends the agreed payment. Here, the antecedent is also a conditional commitment.

To keep CTL^c propositional, actions, missing in our previous work (El-Kholy et al., 2013), are abstracted as predicate propositions that are true at social states precisely after performing agents' local actions.

Example 4.4. Fig. 2 illustrates that the predicate proposition $p = Cancel(Mer. CC(Mer. Cus. pay. goods)))$ is true at the accessible state $s'$ after performing the cancel local action by Mer at the local state $s_{mer}$ to withdraw his commitment $C = CC(Mer. Cus. pay. goods)$ holding at $s$.

Example 4.5. From the purchase protocol described in Section 3.2, we suppose that $CC(Cus. Mer. T. pPrice)$ holds at $s$. When the customer sends the delegateMsg message in order to delegate his commitment to the bank, the predicate proposition $p = Delegate(Cus. Bank. CC(Cus. Mer. T. pPrice))$ will hold at the accessible state $s'$. As we mentioned, the delegation action allows us to formally capture the engagements between three different multi-agent based web services. Technically, these engagements are defined in the protocol by capturing the impact of this message on the active commitment, which is deactivating $CC(Cus. Mer. T. pPrice)$ and activating $CC(Bank. Mer. T. pPrice)$.

Our logical language indeed captures the temporal requirements on the satisfaction of commitments' antecedent and consequent (see Example 4.1). It also captures temporal requirements among different commitments in an orthogonal direction using CTL^c temporal operators or using Boolean connectives in the case that such commitments do not require temporal ordering.

4.1. Semantics of CTL^c formulae

Given the model $M$, the satisfaction of CTL^c formula $\phi$ at a social state $s$ is denoted by $(M, s) \models \phi$. Excluding the commitments and their implications, the semantics of CTL^c state formulae is defined in the model $M$ as usual (semantics of CTL, see for example Clarke et al., 1999). The state formula $CC(i, j, \psi, \phi)$ is satisfied in the model $M$ at $s$ if there is at least one accessible state satisfying the antecedent $\psi$ and the consequent $\phi$ holds in every state satisfying $\psi$ and accessible via $\approx_{ij}$.

$$\neg(M, s) \models CC(i, j, \psi, \phi)\text{ iff } \exists s' \in S_{s.t.s.s_{ij}^{\approx i}}\text{ and } (M, s') \models \psi \text{ and } (2)\forall s' \in S_{s.t.s.s_{ij}^{\approx i}}\text{ and } (M, s') \models \psi, \text{ we have } (M, s') \models \phi$$

Like Singh’s semantics (Singh, 2008), this semantics emphasizes that the antecedent should be satisfied before the consequent satisfaction or at least they are satisfied in the same state, which is strongly required in business contracts (imagine the problem that a bidder does not honor a payment commitment for an auction that it has won). The formula $Fu(CC(i, j, \psi, \phi))$ is satisfied in the model $M$ at $s$ if $s$ satisfies the antecedent $\psi$ and the negation of the commitment $CC(i, j, \psi, \phi)$ and there exists a state $s'$ satisfying the commitment from which $s$ is seen via $\approx_{ij}$.

$$\neg(M, s) \models Fu(CC(i, j, \psi, \phi))\text{ iff } \exists s' \in S_{s.t.s.s_{ij}^{\approx i}}\text{ and } (M, s') \models CC(i, j, \psi, \phi) \text{ and } (M, s) \models \psi \land \neg CC(i, j, \psi, \phi)$$

The idea is that a commitment is fulfilled when we reach an accessible state from the commitment state in which the antecedent holds—as whenever $\psi$ holds in an accessible state, then $\phi$ holds as well—and the commitment becomes no longer active. By stressing that the active commitment should be terminated in the present state (i.e., the fulfillment state), we address the fulfillment paradox appeared in our previous work (Bentahar et al., 2012; El-Menshawy et al., 2013). The violation of commitment can be expressed as follows:

$$EF(CC(i, j, \psi, \phi) \land AG(\neg Fu(CC(i, j, \psi, \phi))))$$

which means that the violation comes out when after having a commitment, the fulfillment does not occur in all states of every possible path.

We conclude this section with the following remark and table.

Remark 4.1.

1. CTL^c formulae cannot be transformed into CTL formulae. For example, to transform the commitment formula, we need first to transform accessibility relations into labeled transitions, which readily need actions, we need first to transform accessibility relations into labeled transitions in order to distinguish them from system (temporal) transitions. Because the standard CTL models do not support labeled transitions, which readily need actions, we conclude that CTL^c is more expressive than CTL.

2. From semantic perspective, unconditional commitment $C(i, j, \phi)$ can be obtained as abbreviation: $C(i, j, \phi) \equiv CC(i, j, T, \phi)$.

3. One can define conditional commitment as follows:

$$CC(i, j, \psi, \phi) \equiv C(i, j, \psi \land \phi) \land \neg C(i, j, \neg \psi)$$

- We prefer to use conditional commitment, a universal frame for commitments, as a first class citizen like recent approaches (see for example Chopra & Singh, 2009; Singh, 2008) and unconditional commitment as a special case like item 2.

- The conditional commitment formula is succinct. In other terms, the conditional commitment formula $CC(i, j, \psi, \phi)$ has a longer equivalent unconditional commitment formula $C(i, j, \psi \land \phi) \land \neg C(i, j, \neg \phi)$. For $n$ conjuncts, $CC(i, j, \psi_1 \land \psi_2 \land \ldots \land \psi_n, \phi_1 \land \phi_2 \land \ldots \land \phi_n)$, the translation into unconditional commitment formula is $2n + 4$. 
We experimentally prove the symbolic algorithm of CTL^c is faster than the one of CTLc introduced in our previous work (Bentahar et al., 2012) (see Section 6).

We introduced in our previous work 14 properties to reason about commitments and their fulfillments (El-Kholy et al., 2013). These properties are still valid here. For example, when a commitment is fulfilled, its consequence holds, and the commitment is no longer active: 
\[
F_iu(CC(i,j,\psi, \phi)) \supset \phi, \quad \text{and} \quad F_iu(CC(i,j,\psi, \phi)) \supset \neg CC(i,j,\psi, \phi).
\]
Also, an agent cannot commit to false: 
\[
\neg F_iu(CC(i,j,\psi, \bot))
\]
Consequently, a commitment to false cannot be fulfilled: 
\[
 \neg F_iu(CC(i,j,\psi, \bot))
\]
Table 1 depicts the formal specification of commitment protocols using the BNF grammar. The symbol “::=” is used to separate choices. The symbols (e.g., Protocol) in the left hand side of “::=” conventionally serve for a nonterminal variables of the grammar. Our protocol specification starts with declaring the name of protocol as an atomic proposition in CTL^c and then listing roles, CTL^c formulae (see Definition 4.5), and messages. The syntactical rule of the message includes the sender, receiver, name, and meaning of the message. The set of message meanings conjoin either commitments, negative commitments, commitment actions or atomic propositions. The antecedents and consequents of commitments are defined as CTL^c formulae. This specification is exemplified in Section 3.2. It specifically extends and improves the specification language of protocols introduced in El-Menshawy et al. (2013) with roles, messages, and protocol name. Furthermore, Baldoni et al. (2013) added explicitly constraints to regulate commitments and (Gerard & Singh, 2013) considered explicitly guarded conditions to control sending and receiving messages in their specification languages of protocols. Our approach basically considers these requirements in terms of: (1) temporal modalities that allow us to satisfy Baldoni et al.’s constraints; and (2) temporal models of protocols that achieve Gerard and Singh’s guarded conditions. This makes our specification language is flexible and more general.

5. Symbolic model checking protocols

Symbolic techniques provide a compact representation for the underlying systems. Also, they are moderately better than automata-based techniques in terms of the execution time and the number of interacting agents, as we showed (El-Menshawy et al., 2013). Conventionally, symbolic techniques alleviate the state explosion problem (see for example Clarke et al., 1999), but cannot eliminate it totally as the state space still increases when the model is getting larger. Given a protocol model M capturing the social behaviors of services composition and a CTL^c formula \( \phi \) expressing a protocol property, the problem of symbolic model checking CTL^c in a nutshell is determining whether or not M is a model for \( \phi \), i.e., can the protocol model M meet the property \( \phi \)? This problem is formalized as \( 〈M, s⟩ \models \phi \) for all \( s \in I \).

5.1. Symbolic algorithm

In this section, we develop a symbolic algorithm to address this decision problem (see Algorithm 1). It takes the model M and CTL^c formula \( \phi \) as input and returns the set of states satisfying \( \phi \). In what follows, we denote the set of states satisfying the formula \( \phi \) by \( 〈\phi〉 \), i.e., \( 〈\phi〉 = \{s \in S(M, s) \models \phi \} \). With respect to the standard CTL algorithm (Clarke et al., 1999), the developed algorithm operates recursively on the structure of \( \phi \) and builds the set \( 〈\phi〉 \). In the basic cases (i.e., when \( \phi \) is an atomic proposition (line 1) and has Boolean connectives: negation and disjunction (line 2 and 3), the algorithm directly returns the set \( 〈\phi〉 \) of states satisfying these cases. In lines 4 to 6, the algorithm calls the standard CTL procedures \( SMC_{EX}(\varphi, M) \), \( SMC_{ES}(\varphi_1, \varphi_2, M) \), and \( SMC_{CC}(\varphi, M) \) introduced in Clarke et al. (1999). In lines 7 to 8, it calls our procedures reported in Algorithms 3 and 4.

Algorithm 1. \( SMC(\varphi, M) \): the set \( 〈\varphi〉 \)

1: \( \varphi \) is an atomic formula: return \( v(\varphi) \);
2: \( \varphi = \neg \varphi_i \): return \( S - SMC(\varphi_i, M) \);
3: \( \varphi = \varphi_i \lor \varphi_j \): return \( SMC(\varphi_i, M) \cup SMC(\varphi_j, M) \);
4: \( \varphi = E\varphi_i \): return \( SMC_E(\varphi_i, M) \);
5: \( \varphi = E(\varphi_i, U\varphi_j) \): return \( SMC_E(\varphi_i, \varphi_j, M) \);
6: \( \varphi = E(\varphi_i, D\varphi_j) \): return \( SMC_E(\varphi_i, \varphi_j, M) \);
7: \( \varphi = CC(i,j,\varphi_1, \varphi_2) \): return \( SMC_{CC}(i,j,\varphi_1, \varphi_2, M) \);
8: \( \varphi = Fu(CC(i,j,\varphi_1, \varphi_2, M)) \):

The procedure \( SMC_{EX}(i,j,s', M) \) in Algorithm 2 computes the set \( Pre_s'(s) \) of pre-images that can see \( s' \) through the accessibility relation \( \equiv_{i,j} \), i.e., \( Pre_s'(s) = \{s \in S_{i,j} \mid s \equiv_{i,j} s'\} \). In a fact of matter, such an algorithm is invoked in Algorithms 3 and 4. Further, since Algorithm 2 directly implements the four conditions in the definition of \( \equiv_{i,j} \), its soundness is straightforward.

Algorithm 2. \( SMC_{EX}(i,j,s', M) \): the set \( Pre_s'(s) \)

1: \( X = \{s \in S \mid l_i(s) = l_i(s') \land \text{ and } s' \in T \land \text{ Var} \lor \text{ Var} = \emptyset \land I^o_i(s) = I^o_i(s') \land \forall v \in \text{ Var} \lor \text{ Var} \}
2: return \( X \);

Algorithm 3 begins by computing the sets X and Y; then proceeds to build the set Z (resp. W) of the states that can see by means of \( \equiv_{i,j} \) a state \( s' \) satisfying \( \psi \) (resp. \( \neg \psi \)). Finally, it returns \( Z - W \), which in turn is the set of states having accessible states satisfying the antecedent and consequence. The soundness of this algorithm is direct from the semantics. For what concerns, suppose \( \psi = CC(i,j,CC(i,j,\varphi_1, \varphi_2), \varphi_3) \). By checking the structure of the formula \( \varphi \), the main algorithm \( SMC(\varphi, M) \) will call \( SMC_{EX}(i,j,\varphi, \varphi_3) \) which \( \psi = CC(i,j,\varphi_1, \varphi_2) \). According to the first step in Algorithm 3, the main algorithm is called with \( \psi = CC(i,j,\varphi_1, \varphi_2) \), which in turn calls \( SMC_{EX}(i,j,\varphi_1, \varphi_2, M) \). When the set of states satisfying \( CC(i,j,\varphi_1, \varphi_2, M) \) is specifically stored in \( X \); so the algorithm proceeds to compute the required sets \( Y \), \( Z \), and \( W \), and then returns the set \( CC(i,j,CC(i,j,\varphi_1, \varphi_2), \varphi_3) \).

Algorithm 3. \( SMC_{EX}(i,j,\psi, \varphi, M) \): the set \( [CC(i,j,\psi, \varphi)] \)

1: \( X = SMC(\psi, M) \);
2: \( Y = SMC(\neg \varphi, M) \);
3: \( Z = \{s \in S \mid \exists s' \in X \text{ s.t. } s \in SMC_{EX}(i,j,s', M) \} \);
4: \( W = \{s \in S \mid \exists s' \in X \cap Y \text{ s.t. } s \in SMC_{EX}(i,j,s', M) \} \);
5: return \( Z - W \);

The procedure \( SMC_{CC}(i,j,\psi, \varphi, M) \) in Algorithm 4 starts by calculating the set \( X \) of states satisfying the commitment \( CC(i,j,\psi, \varphi) \). It then constructs the set \( Y \) of states that satisfy \( \psi \) and \( \neg CC(i,j,\psi, \varphi) \). Afterwards, the algorithm returns the states in \( Y \) that can be seen from a state in \( X \) by means of \( \equiv_{i,j} \). As Algorithm 4 comes directly from the semantics; so its soundness follows:

Algorithm 4. \( SMC_{CC}(i,j,\psi, \varphi, M) \): the set \( (CC(i,j,\psi, \varphi)] \)

1: \( X = SMC(\psi, M) \);
2: \( Y = SMC(\neg \varphi, M) \);
3: \( Z = \{s \in S \mid \exists s' \in X \text{ s.t. } s \in SMC_{EX}(i,j,s', M) \} \);
4: \( W = \{s \in S \mid \exists s' \in X \cap Y \text{ s.t. } s \in SMC_{EX}(i,j,s', M) \} \);
5: return \( Z - W \);

The procedure \( SMC_{CC}(i,j,\psi, \varphi, M) \) in Algorithm 4 starts by calculating the set \( X \) of states satisfying the commitment \( CC(i,j,\psi, \varphi) \). It then constructs the set \( Y \) of states that satisfy \( \psi \) and \( \neg CC(i,j,\psi, \varphi) \). Afterwards, the algorithm returns the states in \( Y \) that can be seen from a state in \( X \) by means of \( \equiv_{i,j} \). As Algorithm 4 comes directly from the semantics; so its soundness follows:
Algorithm 4. SMC\textsubscript{cc}(i,j,ψ,φ,M): the set \([\text{Fu}(CC(i,j,ψ,φ))]\)

1: \(X ← SMC\textsubscript{cc}(i,j,ψ,φ,M);\)
2: \(Y ← SMC(ψ,M) \cap (S − X);\)
3: \(Z ← \{s ∈ Y | ∃s′ ∈ X ∩ SMC\textsubscript{cc}(i,j,s,M)\};\)
4: return \(Z;\)

5.2. Complexity analysis

In this section, we analyze the time and space complexity of model checking CTL\textsuperscript{cc}.

5.2.1. Time complexity

We will prove that the time complexity of model checking CTL\textsuperscript{cc}

is P-complete for explicit models wherein states and transitions are listed explicitly. That is, our algorithm needs a polynomial time to decide whether or not CTL\textsuperscript{cc} formula is correct with respect to
the size of the model and the length of the formula.

Theorem 5.1. The model checking problem for CTL\textsuperscript{cc} can be solved in time \(O(|M| \times |\varphi|)\) where \(|M|\) and \(|\varphi|\) are the size of the explicit model and the length of the formula.

Proof. CTL\textsuperscript{cc} extends CTL, and it is known from (Clarke et al., 1999) that the CTL model checking problem for explicit models is linear in the size of the model and the length of the formula. Thus, we just need to analyze the time complexity of Algorithms 3 and 4. Steps 2 and 3 in these algorithms are simple and it is easy to see that they can be done in linear running time in the size of the model as they are simply constructing sets by performing comparison operations on states. The same argument is valid in case of Algorithm 2. Step 1 in Algorithm 4 calls the model checking procedure recursively on the subformulas \(ψ_1\) and \(ψ_2\) of the formula \(φ = CC(i,j,ψ_1,ψ_2)\). Technically, Algorithm 3 is recursively called till a CTL subformula is encountered. Thus, the depth of the recursion is bounded by the length of the formula \(|φ|\) (i.e., linear in the length \(|φ|\), measured as the number of elements in the closure of \(φ\)). As again model checking CTL is linear in both the size of the model and length of the formula, we conclude that those algorithms have the same complexity. As Algorithm 4 is simply calling Algorithm 3, the result follows. □

Theorem 5.2 (Time complexity of CTL\textsuperscript{cc}). The model checking problem for CTL\textsuperscript{cc} is P-complete for explicit models.

Proof. Membership in P (i.e., upper bound) follows from Theorem 5.1.

Hardness in P (i.e., lower bound) follows by a logspace reduction from model checking CTL proved to be P-complete in Schnoebelen (2002). □

5.2.2. Space complexity

We will prove that the space complexity of model checking CTL\textsuperscript{cc} is PSPACE-complete for concurrent programs. In principle, the class of PSPACE-complete complexity means that the size of the components of concurrent programs needs a polynomial space of the memory usage. The motivation behind considering the complexity of our model checking algorithm for concurrent programs is that explicit representations (e.g., Kripke-like structures as our models in Definition 4.4) are not practicable in practice by literal model checking tools. In a matter of fact, practical model checkers such as MCMAS (for CTL), NuSMV (for CTL and LTL), SPIN (for CTL), and CWB-NC (for CTL\textsuperscript{l}) have the flavor of succinct model languages that disagree upon details and provide the tool with a relatively high-level method of defining concurrent programs. In general, the relation between explicit models and concurrent programs is obtained either by constructing the “reachability graph of concurrent programs” or by computing the product of the components \(P_i\) of concurrent programs (El-Menshawy et al., 2013). Thus, the size of explicit models is exponential in the size of components \(P_i\).

The idea of proof is to utilize a reduction technique to compute the lower and upper bounds of the CTL\textsuperscript{cc} model checking problem. Specifically, our reduction technique is performed in two steps. In the former step, we transform the model of CTL\textsuperscript{cc} into the model of GCTL\textsuperscript{l} in a logspace reduction. We elected the GCTL\textsuperscript{l} model because this model is characterized by labeled transitions with actions and during the transformation process, those labeled transitions are utilized to capture the accessibility relations. The latter step is to transform the GCTL\textsuperscript{l} formulae into the GCTL\textsuperscript{l} formulae in a polynomial space reduction. Before that, we start with briefly reviewing GCTL\textsuperscript{l}, a branching time logic that extends CTL\textsuperscript{l} with action formulae. We then present the transformation procedure and the complexity of the model checking problem. The following BNF grammar defines the syntax of GCTL\textsuperscript{l} (see El-Menshawy et al., 2013 for more details):

\[
S ::= p \mid S \lor S \mid [P]S \mid F[S] \mid G[S] \\
\mathcal{P} ::= 0 \mid (\mathcal{P} \lor \mathcal{P}) \lor \mathcal{P} \\
\text{where } p \in \Phi_x, \Phi_y \text{ is a set of atomic propositions, and } \theta \in \Phi_x, \Phi_y \text{ is a set of atomic action propositions. In this syntax, there are two kinds of formulae: (1) state formulae } S \text{ that hold on a given state; and (2) path formulae } \mathcal{P} \text{ that express temporal properties of paths. The state formulae are the legal formulae of GCTL\textsuperscript{l}. The model of GCTL\textsuperscript{l} is defined as follows.}
\]

Definition 5.1 (Model of GCTL\textsuperscript{l}). A model \(M_G = (S_G, Ac, I_S, I_{Ac}, I_{Ac}, I_G)\) is a tuple where:

- \(S_G\) is a nonempty set of states.
- \(Ac\) is a set of actions.
- \(I_S : S_G \to 2^{\Phi_x}\) is a state labeling function.
- \(I_{Ac} : Ac \to 2^{\Phi_y}\) is an action labeling function.
- \(\rightarrow \subseteq S_G \times Ac \times S_G\) is a labeled transition relation.
- \(I_G \subseteq S_G\) is a set of initial states.

In an intuitive manner, \(S_G\) contains the reachable states of the system, and \(Ac\) the atomic actions that the system might carry out. In this sense, the labeling functions \(I_S\) and \(I_{Ac}\) show which atomic propositions are true in a given state and action respectively. The GCTL\textsuperscript{l} semantics adopts the standard convention employed in temporal logic, such as CTL\textsuperscript{l}. A state satisfies \(A\varphi\) (resp. \(E\varphi\)) if every path (resp. some paths) starting from the state satisfies \(\varphi\). A path satisfies a state formula if the initial state in the path satisfies the formula, and a path satisfies \(\theta\) if the label of the first transition on the path do satisfy \(\theta\). \(X\) represents the “next-time operator” and has the usual semantics. \(\varphi\land \varphi\) holds of a path if \(\varphi\) remains true until \(\varphi\) becomes true. Other temporal modalities are abbreviated as usual. For example, \(F\varphi \equiv \top \land \varphi\) and \(G\varphi \equiv \neg F\neg\varphi\). Also, the universal path quantifier \(A\) can be defined by the existential quantification and negation: \(\neg F\varphi \equiv A\neg\neg F\neg\varphi\).

Our transformation procedure from the problem of model checking CTL\textsuperscript{cc} to the problem of model checking GCTL\textsuperscript{l} is defined as follows: Given a CTL\textsuperscript{cc} model \(M = (S, I, T, \{\pi_{ij}(i,j) \in \mathcal{A}_G^2\}, V)\) and a CTL\textsuperscript{cc} formula \(\varphi\), we have to define a GCTL\textsuperscript{l} model \(M_G = \{F\varphi\}\) and a GCTL\textsuperscript{l} formula \(\mathcal{F}(\varphi)\) using a transformation function \(\mathcal{F}\) such that \(M \models \varphi\) if \(\mathcal{F}(M) \models \mathcal{F}(\varphi)\). The model \(\mathcal{F}(M)\) is defined as a GCTL\textsuperscript{l} model \(M_G = (S_G, Ac, I_S, I_{Ac}, I_G)\) as follows:
1. \( S_c = S, \ I_c = I, \ I_b = \emptyset \),

2. To define the set \( A_c \) of actions, let us first define the set of atomic action propositions \( \Phi_0 = \{ e, x_1, x_2, \ldots, x_n \} \cup \{ p_1, p_2, \ldots, p_m \} \), then \( A_c = \{ x^0, x^1, x^2, \ldots, x^{mn} \} \cup \{ p_1, p_2, \ldots, p_m \} \) where \( x^0 \) and \( x^k \) are the actions labeling transitions respectively defined from the transition relation \( T \) and the accessibility relation \( \approx \) to the polynomial-space reduction. \( \text{CTL} \) and the length of the formula.

^ pertains to the polynomial-space reduction. \( \text{CTL} \) and the length of the formula.

^ depends one-by-one as follows:

(a) \( x^0 \in A_c \), then \( \nu(x^0) = \nu_e \).

(b) \( \nu(x^k) = \nu_{ij} \) for \( 1 \leq i \leq m \) and \( 1 \leq j \leq m \).

(c) \( \nu(p) = \nu_{ij} \) for \( 1 \leq i \leq m \) and \( 1 \leq j \leq m \).

4. The labeled transition relation \( \longrightarrow \) combines the temporal labeled transition \( \longrightarrow \) and the accessibility relations \( \approx \) under the following conditions: for states \( s, s' \in S \),

(a) \( (s, x^k, s') \rightsquigarrow (s, s') \in T \),

(b) \( (s, x^k, s') \rightsquigarrow s \approx s' \in s \approx s' \),

(c) \( (s, p, s') \rightsquigarrow s \approx s' \) if \( s, s' \approx s, s' \).

Let us now define \( \varphi \) as a GCTL* formula by induction on the form of the CTL* formula \( \psi \).

1. \( \varphi(p) = p \), if \( p \) is an atomic proposition;

2. \( \varphi(\neg \varphi) = \neg \varphi(\psi) \);

3. \( \varphi(\varphi \land \psi) = \varphi(\psi) \land \varphi(\psi) \); \( \varphi(\psi) \lor \varphi(\psi) \); \( \varphi(\psi) \lor \varphi(\psi) \);

4. \( \varphi(E \varphi) = EE \varphi(\psi) \);

5. \( \varphi(E \varphi U \varphi) = E(\varphi(\psi) U \varphi(\psi)) \);

6. \( \varphi(E \varphi(\psi)) = E \varphi(\psi) \);

7. \( \varphi CC(i, j, \varphi) = E(a_{ij} \land X \varphi(i)) \land A(i, j, X \varphi(i) \lor X \varphi(j)) \);

8. \( \varphi Fu(C(i, j, \varphi, \psi)) = E(0_{ij} \land X \varphi CC(i, j, \varphi, \psi)) \land \varphi \)

Theorem 5.3 (Soundness of \( \varphi \)). Let \( M \) and \( \varphi \) be respectively a CTL* model and formula and let \( \varphi(M) = \varphi(M) \). We have \( M \models \varphi \) iff \( \varphi(M) \models \varphi(M) \).

Proof. The proof of this theorem is straightforward using the induction proof with respect to the structure of the formula \( \varphi \).

Lemma 5.1. Let \( \models \) pertain to the polynomial-space reduction. \( \text{CTL} \models \text{GCTL} \).

Proof. \( \text{CTL} \) subsumes \( \text{CTL} \) as \( \text{CTL} \) incorporates \( \text{CTL} \) modalities with commitment and fulfillment modalities. Accordingly, \( \text{CTL} \) formulae are also \( \text{CTL} \) formulae and models of \( \text{CTL} \) are already models of \( \text{CTL} \). The transformation of the formula and the model could be readily implemented by a deterministic Turing Machine \( TM \) in a polynomial space w.r.t. the size of the CTL formula, and logarithmic space w.r.t. the size \( n \) of the input CTL model (i.e., \( O(\log n) \)). For the model, the \( TM \) machine simply looks at the input and writes in its output tape, one-by-one, the states (including the initial ones), valuation function, and transitions. For the formula, simply the same input CTL formula is produced as output.

Lemma 5.2. Let \( \models \) be the logspace reduction and let \( \text{Mod}(\text{CTL}) \) and \( \text{Mod}(\text{GCTL}) \) be respectively the model of \( \text{CTL} \) and \( \text{GCTL} \). \( \text{Mod}(\text{CTL}) \models \text{Mod}(\text{GCTL}) \).

Proof. Here we only need to show that the model transformation presented above can be calculated by the \( TM \) machine in space \( O(\log n) \) where \( n \) is the size of the input CTL model. Specifically, \( TM \) reads in the input tape a model of \( \text{CTL} \) and produces in the output tape, one-by-one, the same states (\( S_c = S \)) including the initial ones (\( I_c = I \)) and the same state valuation function (\( I_b = \emptyset \)) as the input. Moreover, \( TM \) writes \( x^k \) in the set of actions \( A_c \) if there are transitions defined in \( T \), the transition relation in the model of \( \text{CTL} \), and reads the accessibility relations \( \approx \) in the input model one-by-one and for each one, it writes \( p \) and \( \beta \) in \( A_c \). Then, for each element in \( A_c \), \( TM \) writes in the output tape, \( \nu(x^k, \beta) \) one-by-one as explained above. Finally, \( TM \) looks at each transition \( (s, s') \) in the input model and writes, one-by-one, the transitions \( (s, x^k, s') \). Similarly, \( TM \) writes, one-by-one, the transitions \( (s, p, s') \) and \( (s, \beta, s') \) for each accessibility relation \( s \approx s' \) in the input model. All these writing operations are clearly logarithmic in space.

Lemma 5.3. Let \( \models \) denote the polynomial-space reduction. \( \text{CTL} \models \text{GCTL} \).

Proof. In Lemma 5.2, we proved that the model of \( \text{CTL} \) is logspace reducible to the model of \( \text{GCTL} \). We also showed above that any formula of \( \text{CTL} \) is transformable into a formula of \( \text{GCTL} \). All those transformations are clearly polynomial in the size of the input formula.

Theorem 5.4 (Space complexity of \( \text{CTL} \)). The complexity of \( \text{CTL} \) model checking for concurrent programs is \( \text{PSPACE} \)-complete with respect to the size of the components \( P \) and the length of the formula.

Proof. From Lemmas 5.1 and 5.3, we proved \( \models \). \( \text{CTL} \) \( \models \text{GCTL} \). The theorem follows from the fact that model checking \( \text{CTL} \) (Schnoebelen, 2002) and \( \text{GCTL} \) (Theorem 6 in El-Menshawy et al. (2013)) are both \( \text{PSPACE} \)-complete for concurrent programs with respect to the size of the components \( P \) and the length of the formula being checked.

In summary, we can argue that although \( \text{CTL} \) extends \( \text{CTL} \), their model checking algorithms still have the same time complexity for explicit models, and the same space complexity for concurrent programs.

6. Implementation and experimental results

We implemented our algorithm on top of the MCMAS symbolic model checker (Lomuscio et al., 2009). We first extended the implementation of the specialized system description language ISPL to include shared and unshared variables, and then extended algorithms of \( \text{CTL} \) with a set of algorithms of accessibility relation, commitments and their fulfillments. MCMAS supports the semantics of interpreted systems and \( \text{CTL} \) and performs OBDD operations by means of the efficient CUDU library. Our model checking tool (i.e., model checker) inherits all features of MCMAS. It specifically provides validation by a graphical simulation, and automatic verification of MASs. Additionally, the graphical user interface that is based on Eclipse supports a wide range of features, such as displaying counter and witness examples, editing and tracking the modeled system. In the following, we present the automated extended ISPL model, which reflects the structure of the extended interpreted systems, generated by our verifier tool. For readability, we commented different sections using “- -.”

Agent: Agent

--Beginning of automaton agent section

Vars: --local variables including local

(continued on next page)
end vars

actions = {};

protocol:

end protocol

evolution:

end evolution

end agent

evaluation

end evaluation

init states

end init states

formulae

end formulae

6.1. Experimental results

6.1.1. First case study

To model the purchase protocol introduced in Section 3.2, we encoded the protocol as the environment agent to be accessible by all participating multi-agent based web services, which are encoded in the corresponding agent sections. In the following, we present the encoded ISPL program of the customer service.

Agent customer

vars: c{c0,c1,c2,c3}; local states

a: {a0,a1}; communication channel for merchant

end vars

actions = {c_order,c_delegate,c_null}; local actions

protocol: local protocol

c = c0; c = cl; c = c2; c = c3;

end protocol

evolution: local transitions

end evolution

end agent

From this program, the evolution of the customer transition with source state c0, target state c1, and guarded condition over the shared variable a and joint action a = (c_order, pr_order) is defined as follows:

c = c1 and a = a1 if c = c0 and action = c_order and environment.action = pr_order;
c = c2 if c = cl and action = c_delegate and environment.action = pr_delegate;
c = c3 if c = c2 and merchant.action = m_inform and environment.action = pr_inform;

end evolution

end agent

It is worth mentioning that the guarded conditions are implemented using the predicates in the form of Guard(X; Y) where Y is the new value assigned to X. So far, the domain of Y includes only the enumeration types. Also, we extended the implementation of MCMAS to automatically generate the guarded automaton with shared and unshared variables of the social system (protocol) using the Cartesian product of m multi-agent based web service automata, modeled by M_i, i ∈ AG and synchronized by joint actions (see Definition 4.4).

To check the correctness of the protocol, we defined five properties in the formulae section. The first three properties check the existence of contractual commitments constituting the protocol, defined as follows:

1. \( EF(Cus. Mer. delBook. pPrice) \)
2. \( EF(Cus. Mer. ⊤. pPrice) \)
3. \( EF(Bank. Mer. ⊤. pPrice) \)

The safety (“something bad never happens”) and liveness (“something good will eventually happen”) properties are also expressed to contribute in checking the correctness of the protocol. The bad situation is that the bank agent fulfills his contract by sending the price, but the merchant never sends the receipt:

4. \( AG(¬(Fu(CC(Bank. Mer. ⊤. pPrice) ∧ ¬EFsReceipt)) \)

The good situation is that on all paths in the future when the merchant delivers the book, there is a path in its future the customer unconditionally commits to send the price:

5. \( AF(delBook ∧ EFCC(Cus. Mer. ⊤. pPrice)) \)

The validation process supported by our tool comprises in running the system interactively to check that it functions as intended. Validating the composition of web services at design time is also acknowledged in Yoo, Jeong, and Cho (2010). However, the authors Yoo et al. (2010) used reachability analysis techniques to check the connectivity between single services being composed w.r.t. “input/output data definition, QoS metrics and values” where composite services are modeled using Petri Nets. Fig. 3 depicts the validation process through which the designer can select the required transition and then check the reachable states so as to compare them with its design model. If there is an error, the designer can edit the system to address the error. This process continues until the designer is assuring that the system is effectively working as intended. Because the validation process does not guarantee that the intended system satisfies certain temporal properties, we have to proceed toward the verification process. Fig. 4 shows the automatic verification results in terms of the execution time in seconds, number of reachable states, and memory usage in bytes. We found that all properties are true.

At this point, we have to go forward to test the effectiveness and scalability of the developed tool. We achieved this goal by performing a set of experiments. Table 2 reports only the verification results of the last four experiments. In fact, we redefine the tested properties in a parameterized form. For example, in the last experiment, the second property is redefined as follows:

\[ EF\left(\bigwedge_{i=1}^{17} CC(Cus. Mer_i. ⊤. pPrice_i)\right) \]

which means that there exists a path where 17 customers will commit eventually to send the price to 17 merchants. Notice that the execution time and number of reachable states increase when the number of agents increases.

6.1.2. Second case study

To acquire more analysis and comparison, we adopt the case study specifying the car damage claims of covered policy holders (Bentahar et al., 2012). The policy holder phones the call center to notify the insurance company AGFIL of a new claim. The call center will gather and validate the information, if so he assigns the nearest garage and gives some suggestions about the approved repairers to the policy holder. The accident form is then sent to AGFIL and Lee CS. AGFIL will check whether the received claim is covered or not. In the case of being invalid, Lee CS will be contacted to stop the process. Otherwise, Lee CS will agree upon repair costs
estimated by the approved repairer (if an assessor is not required) or will otherwise appoint an adjustor who will inspect a car or conduct the assessment itself. When the repairs are completed, the repairer will issue an invoice to Lee CS who will check it against the assessable one. Lee CS monthly sends all invoices to AGFIL to finalize them.

According to this business scenario, we have six guarded automata for Insurer (Ins), playing the role of AGFIL, PolicyHolder (Poh), CallCenter (Cac), Repairer (Rep), Assessor (Ass), playing the role of Lee CS, and Adjustor (Adj). We used the generated code to help encode those agent service automata. We do not worry as the tool automatically generates the corresponding global system. By doing so, we employed the validation process to ensure that the

Table 2

<table>
<thead>
<tr>
<th>#Agents</th>
<th>#States</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>28</td>
<td>638</td>
<td>4.015</td>
</tr>
<tr>
<td>32</td>
<td>1278</td>
<td>113.992</td>
</tr>
<tr>
<td>36</td>
<td>2558</td>
<td>668.446</td>
</tr>
<tr>
<td>40</td>
<td>5118</td>
<td>555.332</td>
</tr>
</tbody>
</table>
developed system is working as intended. In the verification process, we have to express a set of properties using CTL. The bad situation in the safety property is defined as follows:

\[ AG(\neg \text{validClaim} \land \neg CC(\text{Rep, Poh, receivedValidClaim, ERepairCar})) \]

which means that the policy holder’s claim is validated by the assessor and already received by the repairer, but the repairer never commits to repair the car. The liveness property states that whenever the assessor estimates the repair cost, then there exists a path in its future the adjuster will commit to inspect the car:

\[ AG(\text{estimateRepairCost} \land \exists EF CC(\text{Adj, Ass, ReqInspection} \land \text{payInsFees, InspectCar})) \]

Further, we checked in our experiments the existence of 13 contractual commitments and their fulfillments. For example, the following properties:

\[ EF(CC(\text{Cac, Poh, reportAccident, gatherInfo})) \]

\[ EF(Fu(CC(\text{Cac, Poh, reportAccident, gatherInfo})) \]

express the existence of a path so that in its future the call center commits towards the policy holder to gather information if he reports an accident, and such a commitment will eventually be fulfilled, respectively. Other properties are omitted for space limit reasons. Table 3 compares our experimental results of AGFIL scenario against 28 properties with the results introduced in Bentahar et al. (2012) by conducting 7 experiments. As we did above, we redefine the properties in a parameterized form. For example, in experiment 7, the following property:

\[ EF\left(\bigwedge_{i=1}^{4} CC(Cac_i, Poh_j, reportAccident_j, gatherInfo_j)\right) \]

means that there is a possibility for 4 call centers to commit toward 19 policy holders to gather information. Other agents in this experiment are 1 Ins, 11 Rep, 7 Ass, and 7 Adj.

From the table (Table 3), the number of reachable states (which reflects state space) increases exponentially when the number of agents increases as we expected. The number of reachable states and execution times in our approach are better than the corresponding ones reported in Bentahar et al. (2012). Fig. 5 illustrates the relationship between the number of agents and execution times reported in Table 3. Since ordered binary decision diagrams (OBDDs) encoding may change from one model to another based on some internal optimization techniques, the execution time is exposed to this change and may decrease when the number of agents increases as it happens in experiment 7 where OBDD encoding the model is less than the one used in experiment 6 in terms of size (see Fig. 5). We can readily conclude that our tool is faster than the tool introduced in our previous work (Bentahar et al., 2012), especially from experiment 7 (i.e., 49 agents). The reasons are mainly resulted from: (1) the added reachability condition in the definition of the accessibility relation \( i ; j \) (condition 2); (2) the added condition \( \neg CC(i, j, k, l) \) in the semantics of the fulfillment modality; and (3) the semantics of conditional commitment uses \( \exists \) instead of \( \forall \) to compute accessible states. Those requirements technically reduce the number of computed states. To this end, our previous approach (Bentahar et al., 2012) considers only unconditional commitments without actions. Also, because the new accessibility relation and fulfillment semantic are different, their verification algorithms are different as well.

7. Discussion and conclusion

7.1. Discussion

The verifier tool we developed in this paper can be successfully used to verify the conformance of intelligent systems against desirable properties at design time with the aim to: (1) prevent such systems from any loss induced by errors in their design; (2) reduce post-development costs; and (3) increase confidence on the safety, efficiency and robustness. Examples of domains where such systems are fruitfully employed include: (1) industry such as our

![Fig. 5. The relationship between the number of agents and execution times.](image-url)
second case study; (2) government or private universities such as ontology-based planning systems developed for e-Courses (Kontopoulos et al., 2008); (3) B2B applications such as web services choreography approach that integrates effectively B2B processes and allows partners to privately orchestrate their web services (Yeung, 2011); (4) multi-agent systems such as communities of agent-based web services that enable services to collaborate with each other to solve assigned tasks (Khosravifar et al., 2013) and that allow designers to separate the composition functioning of agent-based web services into operational and control behaviors (Bentahar et al., 2013); and (5) business protocols such as our first case study and online auction that allows negotiating the price on the web within a certain period (Lin, Chen, & Chu, 2011). On the other hand, our commitment protocols can find a strong link with expert systems through engineering and coordinating the interactions among components constituting such intelligent systems in a declarative manner. Furthermore, the developed temporal commitment modalities formally model direct interactions among intelligent web services that can be used in those systems.

7.2. Conclusion

The main contribution of this article lies in developing and fully implementing a new verifier tool to automatically check the compositions of multi-agent based web services against a set of properties. Those properties are expressed in CTL$^e$, an extension of CTL with modalities for conditional commitments and their fulfillments along with a suitable extension to consider other commitment actions. In this approach, conditional commitments are used to model the business contractual interactions among services, while commitment protocols are advocated to engineer and regulate those compositions using a declarative formal specification of protocols. Such services and protocols are modeled as guarded automata with variables. We also computed the time and space complexity of model checking CTL$^e$ with regard to explicit models and concurrent programs, which is respectively P-complete and PSPACE-complete. Those complexity results are similar to the ones proved for the standard CTL model checking. Using two case studies, we have experimentally evaluated the effectiveness and efficiency of our symbolic algorithm employed in our tool and implemented on top of MCMAS. These experiments paint the following picture: the developed automatic verifier tool was able to verify a variety of complex formulae correctly and efficiently within a large industrial case study having approximately 1.170e+08 states, thanks to the symbolic encodings used in MCMAS. When comparing our approach to other available proposals, we found that it is considerably succinct the specifications to be checked and develops a symbolic algorithm that is faster than the existing one.

There are many directions for future work. We plan to transform standard languages (e.g., WS-BPEL and OWL-S) into our guarded automata, which will contribute in advancing the state of the art of services verification, because the current approaches that address this problem only provide partial solutions as showed in the recent survey that discusses the strengths and weaknesses of current approaches of testing and verifying web service-centric systems (Bozkurt, Harman, & Hassoun, 2013). In addition to the approaches (Fu et al., 2004; Lomuscio et al., 2012) that we discussed in details their shortcomings in Section 2.2, (García-Fanjul, de la Riva, & Tuya, 2006) translated WS-BPEL directly into the PROMELA (the input language of the SPIN model checker), instead of transforming it first into an XPath-based guarded automata model as in Fu et al.’s approach (Fu et al., 2004). Furthermore, Huang, Tsai, Paul, and Chen (2005) introduced an approach to verify and validate OWL-S process models by making use of the test cases produced in the model checking process. They then extended the BLAST model checker to handle OWL-S’s concurrency constructs and introduced some enhancements in OWL-S and planning domain definition language (PDDL) to facilitate the process of generating test cases. However, the PDDL (Huang et al., 2005) and LTL (García-Fanjul et al., 2006) specification languages do not provide temporal modalities that can be used to formally model direct interactions among web services as our commitment modalities do. We also plan to transform the OWL-P protocols (Desai et al., 2005) into our specification language after extending it with a set of rules to compose different protocols and then use our tool to automatically verify the correctness of the composition. Furthermore, we plan to investigate the problem of modeling other commitment actions as temporal modalities so as to define formal semantics and then develop their symbolic algorithms.

Acknowledgements

We would like to thank NSERC (Canada), FQSC (Quebec) and Menoufa University (Egypt) for their financial supports.

References


