Model Checking Real-Time Conditional Commitment Logic using Transformation

Mohamed El Menshawy, Jamal Bentahar, Warda El Kholy, Amine Laarej

PII: S0164-1212(17)30312-6
DOI: 10.1016/j.jss.2017.12.042
Reference: JSS 10102

To appear in: *The Journal of Systems & Software*

Received date: 15 May 2017
Revised date: 14 December 2017
Accepted date: 31 December 2017

Highlights

- Developing algorithms transforming model checking RTCTLcc into model checking RTCTL
- Implementing a transformation toolkit engine with user interface on top of NuSMV
- Proving the soundness of the transformation technique
- Analyzing the time and space computational complexity of the RTCTLcc model checking

Model Checking Real-Time Conditional Commitment Logic using Transformation

Mohamed El Menshawy[a,b], Jamal Bentahar[a], Warda El Kholy[a], Amine Laarej[a]

[a]*Faculty of Engineering and Computer Science, Concordia Institute for Information Systems Engineering, Concordia University, Canada*
[b]*Faculty of Computers and Information, Computer Science, Menoufia University, Egypt*

Abstract

A new logical language for real-time conditional commitments called RTCTL^{cc} has been developed by extending the CTL logic with interval bounded until modalities, conditional commitment modalities, and fulfillment modalities. RTCTL^{cc} allows us to express qualitative and quantitative commitment requirements in a convenient way. These requirements can be used to model multi-agent systems (MASs) employed in environments that react properly and timely to events occurring at time instants or within time intervals. However, the timing requirements and behaviors of MASs need an appropriate way to scale and bundle and should be carefully analyzed to ensure their correctness, especially when agents are autonomous. In this paper, we develop transformation algorithms that are fully implemented in a new Java toolkit for automatically transforming the problem of model checking RTCTL^{cc} into the problem of model checking RTCTL (real-time CTL). The toolkit engine is built on top of the NuSMV tool, effectively used to automatically verify and analyze the correctness of real-time distributed systems. We analyzed the time and space computational complexity of the RTCTL^{cc} model checking problem. We proved the soundness and completeness of the transformation technique and experimentally evaluated the validity of the toolkit using a set of business scenarios. Moreover, we added a capability in the toolkit to automatically scale MASs and to bundle requirements in a parametric form. We experimentally evaluated the scalability aspect of our approach using the standard ordering protocol. We further validated the approach using an industrial case study.

Keywords: Real-time, qualitative and quantitative commitment requirements, transformation technique, complexity

Email addresses: `moh_marzok75@yahoo.com` (Mohamed El Menshawy), `bentahar@ciise.concordia.ca` (Jamal Bentahar), `w_elkh@encs.concordia.ca` (Warda El Kholy), `laarej.amine@gmail.com` (Amine Laarej)

1. Introduction

The number of formalisms that purportedly facilitate the specification and modeling of social commitments, commitment actions, and commitment properties for concurrent and reactive multi-agent systems (MASs) has exploded over the last years. First-order approaches and temporal logic approaches are the most famous formalisms that can be effectively used to reason about social notions, specifying and modeling multi-agent interactions. Choosing one of them is determined by (1) the data domains over which MASs under investigation operate [Chesani et al., 2013; Chopra and Singh, 2015], (2) the verification techniques, which can be used to verify the compliance of MASs with specifications [El-Menshawy et al., 2011, 2015; El-Kholy et al., 2014], and (3) the expressive power, which can consider deadline constraints, complex expressions, and nested commitments in an integrated framework [Chopra and Singh, 2015; Torroni et al., 2010].

All the current verification techniques employed in first-order approaches that have been put forward to reason about and verify the correctness of social notions were performed at the run-time phase (see, for example, [Chesani et al., 2013]). Because bugs/faults/errors in real-time reactive MASs can be subtle and are perhaps life-threatening, it is very critical to verify the correctness of these systems at the design-time phase. That is, the design of these systems must meet hard real-time requirements before proceeding to implement them. The consideration of formal verifications at design-time is recently validated in industrial domain. For example, in the complex area of avionics, although the standard DO-178B[1] has been successfully operated without producing fatal errors/bugs in the implementation of given software specifications, the cost of complying with the standard is firmly substantial. This is not only because "projects can spend up to seven times more on verification than on other development activities" but also because the model complexity is going gradually to increase [Moy et al., 2013]. Therefore, the avionics industry has updated DO-178B with a new version termed DO-178C which incorporates a supplement of formal methods carried out at design-time and known as DO-333.

In this article, we are concerned with the temporal logic approaches. Computation tree logic (CTL) and linear temporal logic (LTL) are the main sub-classes in these approaches. Specifically, the LTL-based approach considers time to be a linear sequence, while the CTL-based approach adopts a structured tree time in which some nodes (states) have more than a single successor state. It is known as a fact that CTL and LTL are incomparable expressive wise [Clarke et al., 1999]. On the one

[1]It provides the guidance for objectives, activities, and data required to get a certification for stringent avionics software.

hand, theoretically, the program-complexity of CTL and LTL model checking is the same, NLOGSPACE-complete [Schnoebelen, 2002]. However, the formula-complexity of model checking CTL is only LOGSPACE-complete, while PSPACE-complete for LTL model checking [Schnoebelen, 2002], which makes CTL in this very particular respect more efficient than LTL. In the literature, an extension of LTL that allows us to specify quantitative properties has been proposed under the name of metric temporal logic (MTL). In the MTL logic, the modalities of LTL are enriched with timing constraints [Koymans, 1990]. However, the problem of model checking MTL under the continuous semantics is undecidable [Ouaknine and Worrell, 2008]. Several fragments of MTL have been introduced with widely different semantic models and complexities. The most efficient ones are $\text{MTL}_{0,\infty}$, the metric interval temporal logic (MITL), and the bounded MTL (BMTL). $\text{MTL}_{0,\infty}$ has an efficient PSPACE-complete model checking algorithm [Alur et al., 1996]. However, its expressive power is restricted because it requires the constraining timing interval in any temporal modality to have either left endpoint 0 or right endpoint ∞. MITL is a more general and expressive fragment of MTL and augments the constraining time interval in all temporal modalities with the aim to be non-singular and to prohibit punctual specifications. However, the problem of model checking MITL is EXPSPACE-complete [Ouaknine and Worrell, 2008]. BMTL is the subset of MTL in which all constraining time intervals have finite length. As for MITL, the problem of model checking BMTL is EXPSPACE-complete [Ouaknine and Worrell, 2008].

On the other hand, we adopt the CTL-based approach because it (1) allows us to balance between expressiveness and verification complexity as we showed in [El-Kholy et al., 2014, 2015], and (2) is compatible with the formalism of interpreted systems used usually to model concurrent and reactive multi-agent systems [Fagin et al., 1995]. The idea is to enrich the standard CTL logic with modalities to represent, model, and reason about social conditional commitments and their fulfillment. The resulting commitment language (CTL^{cc}) is a special specification language as it incorporates modalities required to flexibly model interactions among intelligent and autonomous agents that cannot be expressed in pure CTL [El-Kholy et al., 2014, 2015]. The intelligence and autonomy properties refer to the fact that agents are not only reactive, but also proactive and have social abilities and constraints, which are modeled through social commitments. Moreover, since the antecedent and consequence of a commitment are arbitrary CTL^{cc} state formulae, they could be commitment formulae as well, which in turn produces a nested commitment (i.e., a commitment to bring about another commitment if a particular commitment holds).

1.1. Current challenging issues

The main challenge of MASs modeling is how to express time bounded properties for autonomous and interacting multi-agent systems employed in environments that react properly and timely to events occurring at time instants or within time intervals in an intuitive and reasonable way. This challenge raises several sub-challenges that need to be addressed including (1) how to model the autonomy, social interactions and flexibility of agents, (2) how to balance social and behavioral constraints with the aspects of intelligence, autonomy, and flexibility of participating agents in the MAS system, (3) how to model check time bounded properties of MASs, especially when agents are autonomous and have to interact with other agents in a social setting, and (4) how to scale up the MAS system.

Regarding the first challenge, the commitment language CTL^{cc} can model successfully interactions among agents in a flexible way while preserving their intelligence and autonomy, as we mentioned above. For example, when an agent is committed to send the payment, he can send it after or before the ordered goods are delivered as along as it is acceptable in the business level. The autonomy is preserved as an agent is only in charge of his commitments, which he created. However, CTL^{cc} still inherits the main limitation of the pure temporal logics including CTL and LTL. The limitation is due to deficiently lacking the capability of measuring the response time (or elapsed time). This time is essentially related to the timing of the event occurrence and the receiving response. The most common methods used to measure real time constraints in the logic context are (1) maximal distance between an event and the receiving response (simply called time-out or deadline), and (2) exact distance (simply called delay). The qualitative commitment properties principally consider the abstract temporal ordering of events using before and after relations. To clarify and explain this limitation, we present the notation $SCC(i, j, \psi, \varphi)$ to specify a strong conditional commitment from an autonomous agent i to another autonomous agent j that if the antecedent ψ holds, then the consequence φ will hold as well. Now, we consider the following example:

Example 1. *The control software agent in the landing gear system commits to eventually retract gears and close doors to take off the aircraft when the pilot agent has been pushed up the handle and stays up [Boniol and Wiels, 2014].*

The requirement stated in Example 1 can be formally expressed using our previous CTL^{cc} logic as a qualitative commitment property as follows:

$$AG(SCC(ContSoft, Pilot, AG\ PushedUp, AF(RetractGears \wedge CloseDoors)))$$

Clearly, it is not sufficient for the control software to eventually retract gears and close doors along all paths to take off the aircraft because the timing constraint on when the events *RetractGears* and *CloseDoors* can occur relative to the event *PushedUp* should be defined. To address the limitation of CTL^{cc}, we added the bounded until modalities. The resulting real-time commitment language is so-called RTCTL^{cc} [El-Kholy et al., 2015].

Regarding the second challenge, the new modalities are able to express quantitative temporal requirements by adding the timing constraints (e.g., deadlines) on the behavior of reactive systems. The hard real-time constraints are defined as time instants or time intervals. Therefore, the multi-modal RTCTL^{cc} logic balances social and behavioral constraints with the aspects of intelligence, autonomy, and flexibility of participating agents in the MAS system in a faithful and convenient way. This is simply because agents can still fulfill or violate their commitments before the expiration of the predefined deadlines. For example, the new modalities contribute in expressing timing deadlines in the antecedent and consequence properties within the realm of commitments. Therefore, the above software requirement can be defined as quantitative commitment property as follows:

$$AG(SCC(\mathit{ContSoft}, \mathit{Pilot}, AG\ \mathit{PushedUp}, AF^{\leq 15}(\mathit{RetractGears} \wedge \mathit{CloseDoors})))$$

The time interval superscribing the eventuality operator F restricts the events *RetractGears* and *CloseDoors* of the control software agent to occur in no more than 15 time units (for instance seconds) from the occurrence of the event *PushedUp* of the pilot agent. The quantitative requirements can generally help designers of MASs effectively model time-critical systems such as the landing gear system [Boniol and Wiels, 2014] and the health-care system for detecting Parkinson patients [García-Magariño and Navarro, 2016]. While RTCTL^{cc} permits reasoning about both qualitative and quantitative requirements, the MTL, MTL$_{0,\infty}$, MITL and BMTL logics provide only quantitative properties and the MTL$_{0,\infty}$ and MITL logics do not support singular/time instants. Moreover, BMTL is not capable of expressing invariance, a basic safety specification.

1.2. Motivations and contributions

Regarding the third challenge, because agents are autonomous and have to interact with other agents in a social setting and real-time environments, we need to formally analyze their behaviors (bounded and unbounded). Different kinds of formal analysis can be classified into three categories (1) deductive methods such as theorem proving, (2) model checking, and (3) abstract interpretation. We elected the second class where the model checking method explores all possible behaviors of an RTCTL^{cc} model to determine whether an RTCTL^{cc} property is satisfied or not. The first motivation and contribution is to develop a toolkit for automatically transforming the problem of model checking RTCTL^{cc} into the problem of model checking RTCTL, a real-time CTL [Emerson et al., 1992]. RTCTL is an extension of the qualitative CTL logic to deal with different sorts of real-time applications. To meet this aim, we develop two algorithms for transforming RTCTL^{cc} model and formulae and these algorithms are implemented in our toolkit using Java. The toolkit engine is built on top of the NuSMV model checker [Cimatti et al., 2002]. The second motivation and contribution is to analyze the computational complexity of the decision problem of RTCTL^{cc} model checking. Specifically, we compute the time and space complexity of this decision problem with respect to explicit models and concurrent programs, respectively. These two aspects give a meaningful picture of the actual computational difficulty behind the problem. The space complexity of the RTCTL^{cc} model checking problem is PSPACE-complete, but the corresponding ones of MITL and BMTL are both EXPSPACE-complete [Ouaknine and Worrell, 2008], although RTCTL^{cc} supports time instants and can express invariance. The third motivation and contribution is to test and evaluate the effectiveness of our toolkit using a set of business scenarios.

Regarding the fourth challenge, our motivation and contribution is to add a capability in the toolkit to automatically scale MASs. To know the required resources (e.g., memory), we increase the number of agents requesting services from the system and bundle automatically the requirements that can check the behaviors of those agents in a parametric form (see Section 5.3 for more details). The performance of the toolkit regarding the scalability aspect is evaluated through 12 experiments using the standard ordering protocol wherein participating agents are increased from 3 to 66. The last motivation is to check the feasibility of our toolkit in industrial domain. We used a real and industrial case study called aircraft landing gear system [Boniol and Wiels, 2014] to further validate our technique.

This work continues as follows. In Section 2, we present the syntax and semantics of RTCTL^{cc} and some details and motivation examples. In Section 3, we present our transformation technique. In Section 4, we analyze the computational complexity of the RTCTL^{cc} model checking problem. In Section 5, we present the full implementation of our transformation toolkit and experimentally test its effectiveness using a set of business scenarios. In the same section, we show how to add the scalability aspect into our toolkit and to evaluate its performance and feasibility using a widely adopted business protocol and real and industrial case study. In Section 6, we discuss the related work. We conclude and identify future research directions in Section 7.

2. The RTCTL^{cc} logic

2.1. Syntax of RTCTL^{cc}

The direct question is how to integrate timing constraints into an RTCTL^{cc} formula. The literature of real-time systems considered three possible solutions (1) bounded temporal operators, (2) freeze quantification, and (3) explicit clock variable. In our previous work, we used the amendment of Emerson et al. [1992] to the solution of bounded temporal operators. This amendment keeps the qualitative temporal operators alongside quantitative temporal operators. Specifically, the interval timing constraints are superscribed the until operators to bound them; so eventuality modalities will have a known end. For the time domain, we choose the set of positive natural numbers denoted by $\mathbb{N}^+$.

Definition 1 (Syntax of RTCTL^{cc}). *The syntax of RTCTL^{cc} is inductively defined as follows:*

$$\begin{aligned}\varphi ::= \;& p \mid \neg\varphi \mid \varphi \vee \varphi \mid EX\varphi \mid EG\varphi \mid E(\varphi\ U\ \varphi) \mid E(\varphi\ U^{[m..n]}\ \varphi)\\ & \mid A(\varphi\ U^{[m..n]}\ \varphi) \mid CC \mid Fu\\ CC ::= \;& WCC(i,j,\varphi,\varphi) \mid SCC(i,j,\varphi,\varphi)\\ Fu ::= \;& FuW(i, WCC(i,j,\varphi,\varphi)) \mid FuS(i, SCC(i,j,\varphi,\varphi))\end{aligned}$$

where:

- *$p \in \mathcal{PV}$ is an atomic proposition. $\neg$ and $\vee$ are the usual Boolean connectives.*
- *E and A are the existential and universal quantifiers on paths.*
- *X, G and U are CTL path modal connectives standing for "next time", "globally", and "until" respectively.*
- *m and $n \in \mathbb{N}^+$ denote the bounds of time intervals where $m \leq n$.*
- *$U^{[m..n]}$ stands for bounded until. This operator is used to abbreviate other eventuality bounded operators (see Table 1).*
- *i and j are two interacting agents. WCC, SCC, FuW and FuS stand for weak and strong conditional commitments and their fulfillments respectively.*

The syntactic grammar rules of RTCTL^{cc} have in principle four different but integrated parts: propositional part, qualitative part, quantitative part, and communication part.

2.1.1. Propositional part

The propositional part is, in fact, a propositional logic and consists of a set of atomic propositions and a set of Boolean connectives. Propositions are declarative statements that can be evaluated into true or false and represent essentially facts. Each fact is declared using the perfective aspect in the English language. For example, with respect to the NetBill protocol, atomic propositions will represent statements such as "the requested goods have been delivered", "the agreed payment has been made", and so on. We can also form more complex statements according to the Boolean connective rules. The following Boolean connectives can be abbreviated in terms of the above as usual: $\wedge$ for conjunction, $\Rightarrow$ for implication, $\equiv$ for equivalence, and $\top$ for the constant true proposition. Because performing actions is the basic method to make communication among agents, we introduce the following schema to represent the atomic propositions: `Noun(key*,action)`. The identification key is optional; so we can remove it if we do not need it. An example is $p = Payment(pID, sent)$, meaning that the payment with the key pID has been sent or simply $p = Payment(sent)$.

2.1.2. Qualitative part

The temporal part (1) allows us to represent and reason about temporal qualitative requirements and to reason about the satisfaction of propositions in the past and future modes, and (2) uses the quantifiers to restrict the execution of paths. A survey about the important role of qualitative requirements in developing qualitative optimization techniques used in software engineering practical problems is introduced in [Santhanam, 2016]. The formula EXp is read as there exists a path such that at the next state of the path p holds, EGp is read as there exists a path such that p holds globally along the path, and $E(\neg Item(delivered)\ U\ \neg Item(delivered) \wedge Payment(sent))$ is read as there exists a path such that $\neg Item(delivered) \wedge Payment(sent)$ eventually holds and $\neg Item(delivered)$ continuously holds until then. That is, the payment will not send until the requested goods have been delivered. Table 1 defines the abbreviations of other qualitative operators.

Table 1: Abbreviations of RTCTL^{cc} qualitative operators

$EF\varphi \equiv E(\top\ U\ \varphi)$
$AG\varphi \equiv \neg EF\neg\varphi$
$A(\varphi\ U\ \psi) \equiv \neg E(\neg\varphi\ U\ (\neg\psi \wedge \neg\varphi)) \wedge \neg EG\neg\varphi$
$AF\varphi \equiv A(\top\ U\ \varphi)$
$AX\varphi \equiv \neg EX\neg\varphi$

2.1.3. Quantitative part

The quantitative part bounds the eventuality of qualitative operators. $E(\varphi\ U^{[m..n]}\ \psi)$ (respectively, $A(\varphi\ U^{[m..n]}\ \psi)$) can be read as "there exists a path such that (respectively, along all paths) ψ eventually holds at time instant i within the interval $[m..n]$ and φ continuously holds from m until then". The sequent is some examples of the quantitative part.

Example 2. *Figure 1 illustrates the models of the following three RTCTL^{cc} formulae: $AG^{\leq 3}\ p$, $EF^{\leq 2}\ p$, and $AG^{\leq 2}\ q \wedge AF^{\leq 4}\ p$. The quantitative modalities ($AG^{\leq m}, EF^{\leq m}$, and $AF^{\leq m}$) are defined in Table 2. The timing constraint m in these modalities defines the "maximum number of permitted transitions along a path before" the propositional variables (p and q) hold [Emerson et al., 1992]. For example, $AF^{\leq 4}\ p$ means that p will inevitably occur in at most four time units or steps.*

Example 3. *RTCTL^{cc} can be accurately used to specify time dependencies. The formula $AG(p \Rightarrow AF^{\leq 20}\ q)$ means that p always leads to q in some future moment, but not later than* 20 *time units.*

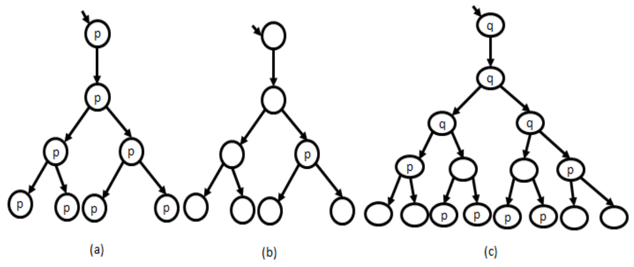


Figure 1: (a) $AG^{\leq 3}\ p$, (b) $EF^{\leq 2}\ p$, and (c) $AG^{\leq 2}\ q \wedge AF^{\leq 4}\ p$

Example 4. *The formula $\neg EF^{\leq 2} PIN(locked)$ means that whenever the PIN is locked, then at least three erroneous tries have been performed. Also, it is not possible to obtain money from the ATM machine in the same session if three mistakes are performed without issuing intermediate reset: $AG\neg(E(\neg Reset(performed) \wedge Error(performed)$ $U^{=3} Money(delivered)))$. By avoiding this situation, the safety property is satisfied in the ATM system.*

Table 2 defines the abbreviations of other quantitative operators. The abbreviated operators alongside basic operators allow us to define complex quantitative requirements. These requirements are relative as they depend on the occurrence of events; instead of using a fixed reference point. Moreover, these requirements inevitably refer to a global notion of time which does not need to be identified with the introduction of a clock. In the table, $EF^{=m}\varphi$ can be read as "there is one possible path through which φ eventually holds exactly at time instant m". Having the bounded opera-

Table 2: Abbreviations of RTCTL^{cc} quantitative operators

$EF^{\leq m}\varphi \equiv E(\top\ U^{\leq m}\ \varphi) \equiv E(\top\ U^{[0..m]}\ \varphi)$
$EF^{=m}\varphi \equiv E(\top\ U^{[m..m]}\varphi)$
$AF^{\leq m}\varphi \equiv A(\top\ U^{\leq m}\ \varphi) \equiv A(\top\ U^{[0..m]}\ \varphi)$
$AF^{[m..n]}\varphi \equiv A(\top\ U^{[m..n]}\varphi)$
$EG^{\leq m}\varphi \equiv \neg AF^{\leq m}\neg\varphi$
$EG^{[m..n]}\varphi \equiv \neg AF^{[m..n]}\neg\varphi$
$AG^{\leq m}\varphi \equiv \neg EF^{\leq m}\neg\varphi$
$AG^{[m..n]}\varphi \equiv \neg EF^{[m..n]}\neg\varphi$
$E(\varphi\ U^{=m}\ \psi) \equiv E(\varphi\ U^{[m..m]}\ \psi)$
$E(\varphi\ U^{\leq 0}\ \psi) \equiv \psi \equiv A(\varphi\ U^{\leq 0}\ \psi)$ [Emerson et al., 1992]

tors held in the model, the corresponding unbounded ones can hold as well [Emerson et al., 1992]. For example, $A(\varphi\ U\ \psi) \equiv \exists m \geq 0\ s.t.\ A(\varphi\ U^{\leq m}\ \psi)$. In this case, there is no formal gain of the expressive power because there are direct translations to get equivalent CTL formulae (e.g., $EX\varphi \equiv EF^{=1}\varphi$). However, when two logics become equally expressive, there is a way to distinguish between them using a succinct property: How each logic can concisely express a given specification [Clarke et al., 2009]. The succinct property is relevant when analyzing the complexity of the model checking problem, because it might influence the size of a formula expressing a given specification property, and thus the time needed to model checking it. The positive result is, bounded operators are exponentially more succinct than CTL. Let us consider the following example:

Example 5. *The formula $EF^{\leq m}p$ allows us to express brief specifications, which would need formulae of exponentially larger size in CTL, formally:*

$$EF^{\leq m}p \equiv p \vee EXp \vee EXEXp \ldots \vee \underbrace{EX \ldots EX}_{m\ times}p$$

2.1.4. Communication part

The communication part focuses on modeling interactions among agents using social commitments and their fulfillments modalities. By using these modalities to express interaction requirements, the resulting properties are called communication properties. The formula $WCC(i,j,\psi,\varphi)$ (respectively, $SCC(i,j,\psi,\varphi)$) is read as "agent i weakly (respectively, strongly) commits towards agent j to consequently satisfy φ once the antecedent ψ holds". Because the antecedent ψ and the consequence φ in the context of commitment modality can be any arbitrary CTL^{cc} formula, so they would be qualitative and/or quantitative formulae. Commitment antecedents can also express the past using the until operator in the usual way. Moreover, the main difference between our two types of conditional commitments (weak and strong) is that an agent can strongly commit only when there is a possibility that the antecedent could be satisfied given that the model is known at design-time, thanks to the fact that the model has finite states. The formula $FuW(i, WCC(i,j,\psi,\varphi))$ (respectively, $FuS(i, SCC(i,j,\psi,\varphi))$) is read as "the weak (respectively, strong) conditional commitment $WCC(i,j,\psi,\varphi)$ (respectively, $SCC(i,j,\psi,\varphi)$) is fulfilled". Examples of this part are as follows:

Example 6. $AG(SCC(ATM, Cus, EF^{\leq 2}PIN(corrected), EF\ Money(delivered)))$ *specifies that the ATM machine always strongly committed to delivering money to the customer when the PIN has been corrected within at most three attempts.*

Example 7. *The customer is strongly devoted to paying the rental amount on the first three days of the rental period:*

$$AG(SCC(Cus, Age, EF\ Car(cID, disposed), EF^{[1..3]}Payment(cID, sent)))$$

According to Chopra and Singh's first-order approach [Chopra and Singh, 2015], this commitment is safe and expressive because the domain of its propositional variables is extensionally quantified along execution paths and these variables are bound by the occurrence of events, which are strongly connected by cID and restricted by the timing interval. Therefore, the payment is expected for this rental car and every payment action refers to a disposed of action via cID.

Example 8. *After one day from disposing of the rental car, the customer sends the agreed payment to fulfill its commitment:*

$$EF\big(FuS(Cus, SCC(Cus, Age, EF\ Car(cID, disposed), EF^{[1..3]} Payment(cID, sent)))\big)$$

The syntax of RTCTL^{cc} also allows us to define nested commitments.

Example 9. *The agency is strongly committed to the customer to withdraw the broken car within two days as soon as the customer commitment is satisfied. In this commitment, the customer is obliged to notify the agency for breaking down at exactly one day when the accident is reported:*

$$AG\Big(SCC(Cus, Age, SCC(Cus, Age, Accident(reported), EF^{=1}\ Notice(cID,$$

$$declared)), EF^{[1..2]}\ Withdraw(cID, broken)\Big)$$

2.2. Semantics of RTCTL^{cc}

To interpret RTCTL^{cc}-formula, we define a logical model. In fact, this model is generated from our extended version of the interpreted system formalism. The formalism of interpreted systems provides a very popular framework to model MASs. Specifically, in [Bentahar et al., 2012; El-Kholy et al., 2014; El-Menshawy et al., 2013], we extended the original formalism introduced in [Fagin et al., 1995] with sets of shared and unshared variables to account for agent communication. The extended version of interpreted systems is composed of a set $\mathcal{A} = \{1, \dots, n\}$ of n autonomous agents plus the environment agent e. Each agent $i \in \mathcal{A}$ is characterized by:

1. A set L_i of finite local states. Each private local state l_i represents the whole information about the system that the agent has at a given moment.
2. A set Act_i of finite local actions available to the agent, including the fulfillment actions and *null* action, which refers to the fact of doing nothing.
3. A set Var_i of at most $n-1$ local variables to model communication channels with all other agents. Through such channels values are sent and received as in distributed systems. Intuitively, $|Var_i| \leq n-1$ as i might not have communication channels with particular agents.

4. A local protocol function $\mathcal{P}_i : L_i \to 2^{Act_i}$, which represents the decision-making procedure of i and produces the set of enabled actions that might be performed by i in a given local state.
5. A set $\iota_i \subseteq L_i$ of initial states.
6. A local transition function $\tau_i : L_i \times Act_i \to L_i$, which defines the evolution from a local state to another local state given a local action a.

The environment agent e, which captures the information that might not pertain to a specific agent, is characterized by $L_e, Var_e, Act_e, \mathcal{P}_e, \iota_e$ and τ_e. The notion of social state (termed global state in [Fagin et al., 1995]) represents the screenshot of all agents in the system at a certain moment. A social state $s \in S$ is a tuple $s = (l_1, \dots, l_n, l_e)$ where each element $l_i \in L_i$ represents the i's local state alongside the environment state l_e. The set of all social states $S \subseteq L_1 \times \dots \times L_n \times L_e$ is a subset of the Cartesian product of all local states of all agents and the environment agent. All local transition functions are combined together to define a social transition function $\tau : S \times ACT \to S$ in order to give the overall transition function for the system where $ACT = Act_1 \times \dots \times Act_n \times Act_e$ is called a joint or shared action (one for each agent and environment agent) and represents a synchronous action of the system as a whole. Let $l_i(s)$ denotes the local state of the agent i in the social state s and the value of a variable x in the set Var_i at $l_i(s)$ is denoted by $l_i^x(s)$. When $l_i(s) = l_i(s')$, then for all $x \in Var_i$ we have $l_i^x(s) = l_i^x(s')$. To allow agent i to communicate with agent j, they should share a channel, which is represented by a shared variable between them. Formally, a communication channel between i and j coexists as long as $|Var_i \cap Var_j| = 1$. For the variable $x \in Var_i \cap Var_j$, $l_i^x(s) = l_j^x(s')$ means that the values of x in $l_i(s)$ for i and in $l_j(s')$ for j are the same. Intuitively, the existence of a communication channel between i in s and j in s' means the value of the variable x has been sent by one of them towards the other, therefore, i and j will have the same value for this variable as a consequence of the communication between them. The valuation function $\mathcal{V} : \mathcal{AP} \to 2^S$ defines what atomic propositions are true from the set $\mathcal{AP}$ at system states. In summary, the extended version of the formalism of interpreted systems is denoted by the tuple $IS^+ = \big(\{L_i, Var_i, Act_i, \mathcal{P}_i, \tau_i, \iota_i\}_{i \in \mathcal{A}}, \{L_e, Var_e, Act_e, \mathcal{P}_e, \tau_e, \iota_e\}, \mathcal{V}\big)$.

Definition 2 (RTCTL^{cc} models)**.** *A model $M = \big(S, I, T, \{\sim_{i \to j} \mid (i,j) \in \mathcal{A}^2\}, \mathcal{V}\big)$ is generated from $IS^+ = \big(\{L_i, Var_i, Act_i, \mathcal{P}_i, \tau_i, \iota_i\}_{i \in \mathcal{A}}, \{L_e, Act_e, \mathcal{P}_e, \tau_e, \iota_e\}, \mathcal{V}\big)$ by synchronising joint actions of $n+1$ agent models as follows:*

– *S is a set of social states for the system.*

– *$I \subseteq \iota_1 \times \dots \times \iota_n \times \iota_e$ is a set of initial states for the system such that $I \subseteq S$.*

- *$T \subseteq S \times S$ is a total temporal relation (i.e., each state has at least one successor) defined by $(s, s') \in T$ iff there is a joint action $a = (a_1, \dots, a_n, a_e) \in ACT$ such that $\tau(s, a_1, \dots, a_n, a_e) = s'$.*
- *For each pair of autonomous agents $(i, j) \in \mathcal{A}^2$, $\sim_{i \to j} \subseteq S \times S$ is a social accessibility relation defined by $s \sim_{i \to j} s'$ iff the following conditions hold (1) $l_i(s) = l_i(s')$, (2) $(s, s') \in T$, (3) $\forall x \in Var_i \cap Var_j$ such that $Var_i \cap Var_j \neq \emptyset$ we have $l_i^x(s) = l_j^x(s')$, and (4) $\forall y \in Var_j - Var_i$ we have $l_j^y(s) = l_j^y(s')$.*
- *$\mathcal{V} : \mathcal{PV} \to 2^S$ is a labeling function defined as in IS^+.*

As in [Emerson et al., 1992], all transitions happen instantaneously in our quantitative temporal model M, i.e., each transition takes a single time unit for execution from a source state to a destination state. So, the duration of a transition can be called either a weight or a cost. The underlying real-time model is discrete and has a tree structure of states. Each tree represents a real-time system model. The model M is unwound into a set of execution paths in which each path $\pi = s_0, s_1, \dots$ is an infinite sequence of social states increasing simultaneously over time such that $s_i \in S$ and $(s_i, s_{i+1}) \in T$ for each $i \geq 0$. $\pi(k)$ is the k-th state of the path π. The set of all paths starting at s is denoted by $\Pi(s)$.

Definition 3 (Semantics of RTCTL^{cc}). *Given the model M, the satisfaction of RTCTL^{cc} formula φ in a state s denoted by $(M, s) \models \varphi$ is recursively defined as follows:*

- *$(M, s) \models p$ iff $s \in \mathcal{V}(p)$,*
- *$(M, s) \models \neg\varphi$ iff $(M, s) \not\models \varphi$,*
- *$(M, s) \models \varphi \vee \varphi$ iff $(M, s) \models \varphi$ or $(M, s) \models \varphi$,*
- *$(M, s) \models EX\varphi$ iff $\exists \pi \in \Pi(s)$ such that $(M, \pi(1)) \models \varphi$,*
- *$(M, s) \models EG\varphi$ iff $\exists \pi \in \Pi(s)$ such that $\forall k \geq 0, (M, \pi(k)) \models \varphi$,*
- *$(M, s) \models E(\varphi\ U\ \psi)$ iff $\exists \pi \in \Pi(s)$ such that $\exists k \geq 0$, $(M, \pi(k)) \models \psi$ and $\forall j, 0 \leq j < k, (M, \pi(j)) \models \varphi$,*
- *$(M, s) \models E(\varphi\ U^{[m..n]}\ \psi)$ iff $\exists \pi \in \Pi(s)$ such that $\exists i, m \leq i \leq n$, $(M, \pi(i)) \models \psi$ and $\forall j, m \leq j < i, (M, \pi(j)) \models \varphi$,*
- *$(M, s) \models A(\varphi\ U^{[m..n]}\ \psi)$ iff $\forall \pi \in \Pi(s)$ such that $\exists i, m \leq i \leq n$, $(M, \pi(i)) \models \psi$ and $\forall j, m \leq j < i, (M, \pi(j)) \models \varphi$,*

- $(M, s) \models WCC(i, j, \psi, \varphi)$ *iff* $\forall s' \in S$ *such that* $s \sim_{i\to j} s'$ *and* $(M, s') \models \psi$, $(M, s') \models \varphi$,
- $(M, s) \models SCC(i, j, \psi, \varphi)$ *iff* (1) $\exists s' \in S$ *such that* $s \sim_{i\to j} s'$ *and* $(M, s') \models \psi$, *and* (2) $(M, s) \models WCC(i, j, \psi, \varphi)$,
- $(M, s) \models FuW(i, WCC(i, j, \psi, \varphi))$ *iff* $\exists s' \in S$ *such that* $s' \sim_{i\to j} s$ *and* $(M, s') \models WCC(i, j, \psi, \varphi)$ *and* $(M, s) \models \varphi \wedge \neg WCC(i, j, \psi, \varphi)$,
- $(M, s) \models FuS(i, SCC(i, j, \psi, \varphi))$ *iff* $\exists s' \in S$ *such that* $s' \sim_{i\to j} s$ *and* $(M, s') \models SCC(i, j, \psi, \varphi)$ *and* $(M, s) \models \psi \wedge \neg SCC(i, j, \psi, \varphi)$.

The meaning of our semantic rules of each modality is introduced in [El-Kholy et al., 2015].

3. Transformation technique to model check RTCTL^{cc}

The decision problem of RTCTL^{cc} model checking is to check whether or not a model representing a real-time MAS M satisfies a property expressed as RTCTL^{cc}-formula φ: Does $(M, s) \models \varphi$ for all $s \in I$? where I is a set of initial social states. In order to address this problem, a transformation has been acknowledged as an alternative technique that transforms the problem of model checking MASs [Lomuscio et al., 2007] and protocols [El-Menshawy et al., 2011] into existing model checking approaches. This transformation technique reveals several advantages. Firstly, it enables designers to use existing model checking tools rather than developing and implementing a new tool to tackle the problem, which is not a straightforward task [El-Menshawy et al., 2011; El-Menshawy et al., 2013]. Secondly, it provides a suitable way to compare between different verification techniques with respect to the same model checking problem [El-Menshawy et al., 2011; El-Menshawy et al., 2013]. Current transformation techniques can be performed formally or informally. El-Menshawy et al. [2013] criticized informal transformation techniques, as there are no formal rules that can be semantically used to prove the correctness of the developed transformation algorithms.

In this section, we present a formal transformation technique to transform the problem of model checking RTCTL^{cc} into the problem of model checking RTCTL [Emerson et al., 1992], so that the NuSMV model checker is feasible [Cimatti et al., 2002]. We selected the branching real-time temporal logic RTCTL for two technical reasons. The first reason is that the RTCTL model is based on the Kripke structure as our model. Therefore, the transformation process will only need a logarithmic space to be performed. The second reason is that RTCTL enables us to use NuSMV to automatically verify the correctness of nested temporal modalities, which other real-time model checkers, such as UPPAAL[2], do not support. The sequent BNF grammar presents the Emerson et al.'s RTCTL syntax [Emerson et al., 1992]:

$$\varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid EX\varphi \mid E(\varphi\ U\ \varphi) \mid EG\varphi \mid EF^{[m..n]}\varphi \mid AF^{[m..n]}\varphi$$
$$\mid EG^{[m..n]}\varphi \mid AG^{[m..n]} \mid A(\varphi\ U^{[m..n]}\ \varphi) \mid E(\varphi\ U^{[m..n]}\ \varphi)$$

where m and $n \in \mathbb{N}^+$ define the timing constraint ranges and $p \in \mathcal{PV}$ is an atomic proposition. From this syntax, RTCTL is CTL plus bounded modalities.

Definition 4 (RTCTL Models)**.** *A model $M_t = (S_t, I_t, T_t, L_t)$ is a tuple where S_t is a nonempty set of states, $I_t \subseteq S_t$ is a set of initial states, $T_t \subseteq S_t \times S_t$ is a transition relation, and $L_t : S_t \to 2^{\mathcal{PV}}$ is a labeling function assigning to each state a set of atomic propositions to interpret this state.*

The semantics of RTCTL formulae are similar to our semantics of RTCTL^{cc}, except the semantics of commitments and their fulfillments. Informally, the semantics of RTCTL bounded modalities are given as follows:

- $EF^{[m..n]}p$ states that there is a path such that p holds in a future time instant i where $m \leq i \leq n$.
- $AF^{[m..n]}p$ states that along all paths, p holds in a future holds in a future time instant i where $m \leq i \leq n$.
- $EG^{[m..n]}p$ states that there is a path such that p holds in all future time instants i where $m \leq i \leq n$.
- $AG^{[m..n]}p$ states that along all paths, p holds in all future time instants i where $m \leq i \leq n$.
- $E(p\ U^{[m..n]}\ q)$ states that there is a path such that q holds in the future time instant i where $m \leq i \leq n$ and p holds in all future time instants j where $m \leq j < i$.
- $A(p\ U^{[m..n]}\ q)$ states that along all paths, q holds in a future time instant i where $m \leq i \leq n$ and p hold in all future time instants j where $m \leq j < i$.

At this point, we are ready to present our transformation technique. It specifically consists of two steps: model transformation and formula transformation. Figure 2 illustrates the workflow of our RTCTL^{cc} model checking process using the transformation technique. Initially, we model the underlying MAS using our model M and formalize the desirable property using RTCTL^{cc}. By doing so, the transformation technique starts. The following sections discuss in details this technique.

[2]`http://www.uppaal.org/`

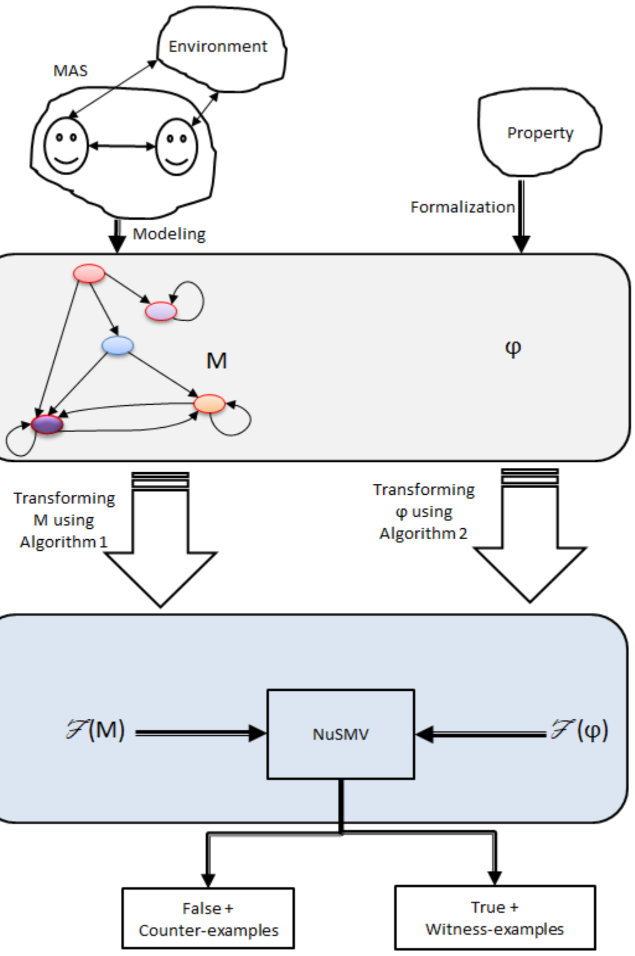


Figure 2: The process of model checking RTCTL^{cc}

3.1. Transforming the RTCTL^{cc} model

In this section, we develop an algorithm that automatically transforms the RTCTL^{cc} model $M = (S, I, T, \{\sim_{i\to j} \mid (i,j) \in \mathcal{A}^2\}, \mathcal{V})$ into the RTCTL model $M_t = (S_t, I_t, T_t, L_t)$ (see Algorithm 1). In lines 1 and 2, it transforms the set S of the system states and the set I of initial states into the corresponding ones. The transformation process of these sets is bijective (one-to-one correspondence). In line 3, the algorithm defines a new set of atomic propositions by conjoining three sets of atomic propositions. The description of each set is introduced in the algorithm. The new set is used by the function L_t to label states. In line 4, the algorithm proceeds to transform the transition relations in T and the asymmetric closures of the accessibility relations to constitute the transition relations in T_t according to two conditions. The first condition checks that if the two states have a transition relation in T, then this transition relation becomes a transition relation in T_t. The second condition checks that if the current state s has an accessible state s' using the accessibility relation, i.e., $s \sim_{i \to j} s'$, then the algorithm checks that (1) if the symmetric closure of the accessibility relation is not in T_t, then a new transition from s' into s is added in T_t, (2) a new set of atomic propositions is added in s' alongside the atomic proposition labeling s' using $\mathcal{V}$ in the model M for the interacting agents i and j in order to distinguish the accessible states s' from s, and (3) the algorithm adds a new set of atomic propositions into the label of s defined by $\mathcal{V}$ in the model M for the interacting agents i and j in order to distinguish the asymmetric closure of the accessibility relation and to capture the semantics of the fulfillment actions.

Algorithm 1 An RTCTL^{cc} model $M = (S, I, T, \{\sim_{i \to j} \mid (i,j) \in \mathcal{A}^2\}, \mathcal{V})$: An RTCTL model $M_t = (S_t, I_t, T_t, L_t)$

```
1: $I_t := I$,
2: $S_t := S$,
3: $L_t : S_t \to 2^{\mathcal{PV}'}$ where $\mathcal{PV}'$ is defined as the union of the following three sets of
   atomic propositions (i.e., $\mathcal{PV}' := \mathcal{PV} \cup X \cup Y$):
     – The set $\mathcal{PV} := \{p, q, \dots\}$ of atomic propositions in the model $M$ to capture
       the semantics of bounded and unbounded modalities.
     – The set $X := \{\alpha^1\alpha^1, \alpha^1\alpha^2, \dots, \alpha^n\alpha^n\}$ for the social accessibility relation
       $\sim_{i \to j}$ to capture the semantics of commitments.
     – The set $Y := \{\beta^1\beta^1, \beta^2\beta^1, \dots, \beta^n\beta^n\}$ for the symmetric closure of the social
       accessibility relation $\sim_{i \to j}$ to capture the semantics of fulfillment modalities.
4: The transition relation $T_t$ combines the temporal transition $T$ and asymmetric
   closures of the accessibility relations under the sequent conditions: for states
   $s, s' \in S$,
     1. If $(s, s') \in T$, then $(s, s') \in T_t$,
     2. If $s \sim_{i \to j} s'$, then:
          – If $(s', s) \notin T_t$, then $T_t := T \cup \{(s', s)\}$,
          – $L_t(s') := \mathcal{V}(s') \cup \{\alpha^i\alpha^j\}$, $1 \le i \le n$ and $1 \le j \le n$, and
          – $L_t(s) := \mathcal{V}(s) \cup \{\beta^i\beta^j\}$, $1 \le i \le n$ and $1 \le j \le n$
```

3.2. Transforming the RTCTL^{cc} formulae

In this section, we develop an algorithm (refer to Algorithm 2) that automatically transforms RTCTL^{cc} formula φ into RTCTL formula $\mathscr{F}(\varphi)$ using a transformation function $\mathscr{F}$. Our transformation function is recursive with respect to the structure of φ. The fragment of CTL in RTCTL^{cc} is transformed directly into the corresponding CTL fragment in RTCTL in lines $1, 2, 3, 4, 5$, and 6. In lines 7 and 8, the algorithm transforms the quantitative formulae into the corresponding ones. In lines $9, 10, 11$, and 12, the algorithm proceeds to transform our communication formulae according to the defined semantics. For example, the weak commitment is transformed into an RTCTL formula stating that along all paths in the next state if the transformation of the antecedent ψ and the atomic proposition added to represent the accessibility relation are true in this state, then the transformation of the consequence φ should hold as well. For the transformation of the two fulfilment formulae, it is important to add in the next state the proposition $\beta^i\beta^j$ to precisely identify the commitment state from which the current state is accessible in the original model and distinguish this next state from any other next state that satisfies the commitment formulae without having accessibility to the current state.

Algorithm 2 A RTCTL^{cc} formula φ: A RTCTL formula $\mathscr{F}(\varphi)$

```
1: $\mathscr{F}(p)=p$, if $p$ is an atomic proposition,
2: $\mathscr{F}(\neg\varphi) = \neg\mathscr{F}(\varphi)$,
3: $\mathscr{F}(\varphi \vee \psi) = \mathscr{F}(\varphi) \vee \mathscr{F}(\psi)$,
4: $\mathscr{F}(EX\varphi) = EX\mathscr{F}(\varphi)$,
5: $\mathscr{F}(E(\varphi\ U\ \psi)) = E(\mathscr{F}(\varphi)\ U\ \mathscr{F}(\psi))$,
6: $\mathscr{F}(EG\varphi) = EG\mathscr{F}(\varphi)$,
7: $\mathscr{F}(E(\varphi\ U^{[m..n]}\ \psi)) = E(\mathscr{F}(\varphi)\ U^{[m..n]}\ \mathscr{F}(\psi))$,
8: $\mathscr{F}(A(\varphi\ U^{[m..n]}\ \psi)) = A(\mathscr{F}(\varphi)\ U^{[m..n]}\ \mathscr{F}(\psi))$,
9: $\mathscr{F}(WCC(i,j,\psi,\varphi)) = AX((\mathscr{F}(\psi) \wedge \alpha^i\alpha^j) \Rightarrow \mathscr{F}(\varphi))$,
10: $\mathscr{F}(SCC(i,j,\psi,\varphi)) = EX(\mathscr{F}(\psi) \wedge \alpha^i\alpha^j) \wedge \mathscr{F}(WCC(i,j,\psi,\varphi))$,
11: $\mathscr{F}(FuW(i, WCC(i,j,\psi,\varphi))) = EX(\beta^i\beta^j \wedge \mathscr{F}(WCC(i,j,\psi,\varphi))) \wedge \mathscr{F}(\varphi) \wedge$
    $\neg\mathscr{F}(WCC(i,j,\psi,\varphi))$,
12: $\mathscr{F}(FuS(i, SCC(i,j,\psi,\varphi))) = EX(\beta^i\beta^j \wedge \mathscr{F}(SCC(i,j,\psi,\varphi))) \wedge \mathscr{F}(\psi) \wedge$
    $\neg\mathscr{F}(SCC(i, j,\psi,\varphi))$.
```

Theorem 1 (Soundness and Completeness of $\mathscr{F}$)**.** *Let M and φ be respectively an RTCTL^{cc} model and a formula and let $\mathscr{F}(M)$ and $\mathscr{F}(\varphi)$ be the corresponding model and formula in RTCTL. We have $M \models \varphi$ iff $\mathscr{F}(M) \models \mathscr{F}(\varphi)$.*

Proof. The proof of this theorem is by induction with respect to the structure of the formula φ.

- For RTCTL formulae, the result is straightforward because $\mathscr{F}(\varphi) = \varphi$.

- For the formula $\varphi = WCC(i, j, \psi, \phi)$, we have $(M, s) \models WCC(i, j, \psi, \phi)$ iff for every $s' \in S$ such that $s \sim_{i \to j} s'$ and $(M, s') \models \psi$, we have $(M, s') \models \phi$. Consequently, from the definition of $\mathscr{F}(M)$ and $\mathscr{F}(\varphi)$, we obtain $(M, s) \models WCC(i, j, \psi, \phi)$ iff for every $s' \in S_t$ such that $(s, s') \in T_t$, $\alpha^i\alpha^j \in L_t(s')$, and $(M_t, s') \models \mathscr{F}(\psi)$, we have $(M_t, s') \models \mathscr{F}(\phi)$. By semantics of AX in RTCTL, we obtain $(M_t, s) \models AX((\mathscr{F}(\psi) \wedge \alpha^i\alpha^j) \Rightarrow \mathscr{F}(\phi))$.

- For the formula $\varphi = SCC(i, j, \psi, \phi)$, we have $(M, s) \models SCC(i, j, \psi, \phi)$ iff (1) there exists $s' \in S$ such that $s \sim_{i \to j} s'$, and $(M, s') \models \psi$, and (2) $(M, s) \models WCC(i, j, \psi, \phi)$. Consequently, from the definition of $\mathscr{F}$, we get $(M, s) \models SCC(i, j, \psi, \phi)$ iff (1) there exists $s' \in S_t$ such that $(s, s') \in T_t$, $\alpha^i\alpha^j \in L_t(s')$, and $(M_t, s') \models \mathscr{F}(\psi)$, and (2) $(M_t, s) \models \mathscr{F}(WCC(i, j, \psi, \phi))$ (which is proved above). By semantics of EX in RTCTL, we obtain $(M_t, s) \models EX(\mathscr{F}(\psi) \wedge \alpha^i\alpha^j) \wedge \mathscr{F}(WCC(i, j, \psi, \phi))$.

- For the formula $\varphi = FuW(i, WCC(i, j, \psi, \phi))$, we have $(M, s') \models FuW(i, WCC(i, j, \psi, \phi))$ iff:

  1. There exists $s \in S$ such that:
     (a) $s \sim_{i \to j} s'$, and
     (b) $(M, s) \models WCC(i, j, \psi, \phi)$, and
  2. $(M, s') \models \phi$, and $(M, s') \not\models WCC(i, j, \psi, \phi)$.

  Consequently, from the definition of $\mathscr{F}$, we get $(M, s') \models FuW(i, WCC(i, j, \psi, \phi))$ iff:

  1. There exists $s \in S_t$ such that:
     (a) $(s', s) \in T_t$ and $\beta^i\beta^j \in L_t(s)$, and
     (b) $(M_t, s) \models \mathscr{F}(WCC(i, j, \psi, \phi))$ (which is proved above), and
  2. $(M_t, s') \models \mathscr{F}(\phi)$, and $(M_t, s') \not\models \mathscr{F}(WCC(i, j, \psi, \phi))$ (which is proved above).

  By semantics of EX in RTCTL, we obtain $(M_t, s') \models EX(\beta^i\beta^j \wedge \mathscr{F}(WCC(i, j, \psi, \phi))) \wedge \mathscr{F}(\phi) \wedge \neg\mathscr{F}(WCC(i, j, \psi, \phi))$.

- For the formula $\varphi = FuS(i, SCC(i, j, \psi, \phi))$, the proof is similar to the case of $\varphi = FuW(i, WCC(i, j, \psi, \phi))$ by simply replacing $\mathscr{F}(WCC(i, j, \psi, \phi))$ and $\mathscr{F}(\phi)$ by $\mathscr{F}(SCC(i, j, \psi, \phi))$ and $\mathscr{F}(\psi)$, respectively, so the theorem.

□

It is worth noticing that an RTCTL^{cc} formula holds in M iff the corresponding RTCTL formula holds in the RTCTL model obtained from M. In fact, we are discussing heretofore the transformation of the model checking problem, meaning that any original RTCTL^{cc} formula φ in the original model M is equivalent to the model checking of the transformed RTCTL formula $\mathscr{F}(\varphi)$ in the transformed model $\mathscr{F}(M)$, which does not entail that any RTCTL^{cc} formula can be expressed in RTCTL.

4. Complexity analysis

Given a model M and an RTCTL^{cc}-formula φ, the complexity of its model checking can be evaluated in terms of the size of the model $|M|$ and the length of the formula $|\varphi|$. The size of the explicit model can be computed by $|M| = |S| + |T|$. The length of φ is: $|\varphi| = |\varphi'| + c$ where $|\varphi'|$ is the length of the formula φ after deleting time bounds and c is the summation of the lengths of the bit strings, which represent the time bounds of φ in a binary format as shown by Emerson et al. [1992]. For example, if $\varphi = EF\ SCC(i, j, AF^{=5}\ p, q)$, then $\varphi' = EF\ SCC(i, j, AF\ p, q)$. So, we have $|\varphi'| = 7$. The time bound 5 is represented by 101 in binary, which is a bit string of length 3. Therefore, $|\varphi| = 7 + 3 = 10$.

In this section, we analyze the time and space complexities of the RTCTL^{cc} model checking problem. As our approach is transformation-based, we start by analyzing the time complexity of transforming the RTCTL^{cc} model and formula with respect to explicit models, where all states and transitions are enumerated. Specifically, we prove that these two transformations are linear with respect to the input RTCTL^{cc} model and formula. For the specific case of the formula transformation, the proof is based on analyzing the size of the obtained RTCTL formula with respect to the size of the input RTCTL^{cc} formula. The idea is to show that the size of the output formula is bounded by the size of the input formula subject to a constant c. The linearity of these two transformations entails the P-completeness of the model checking problem of RTCTL^{cc} in explicit models. Given that, we proceed to analyze the space complexity of the RTCTL^{cc} model checking problem and prove its PSPACE-completeness with respect to concurrent programs where the model has the form of a synchronized product of m programs.

Proposition 1. *The time complexity of transforming the RTCTL^{cc} model is linear with respect to the size of the input model M, i.e., $\mathcal{O}(|M|)$.*

Proof. The proof is straightforward from the model transformation algorithm (see Algorithm 1). Specifically, Algorithm 1 is an on the fly algorithm, which takes the RTCTL^{cc} model M as input and writes one by one the states (including the initial ones), valuation function, accessibility relations, and transitions to produce the corresponding RTCTL model. All these transformation steps take a linear time with respect to the size of the input, so the result. $\square$

Proposition 2. *Let φ be an RTCTL^{cc} formula and $\mathscr{F}$ the transformation function defined in Algorithm 2. There exists a constant c such that $|\mathscr{F}(\varphi)| < c|\varphi|$.*

Proof. The proof is by induction on the structure of the formula.

- For RTCTL formulae, the result is straightforward because $|\mathscr{F}(\varphi)| = |\varphi|$.
- For the formula $\varphi = WCC(i, j, \psi, \phi)$, we have $|\mathscr{F}(\varphi)| = |\mathscr{F}(\psi)| + |\mathscr{F}(\phi)| + 4$. Thus, by assumption that the proposition holds for the formulae ψ and ϕ, $\exists c_1, c_2$ s.t. $|\mathscr{F}(\varphi)| < c_1|\psi| + c_2|\phi| + 4$. Because $|\psi| < |\varphi|$, $|\phi| < |\varphi|$ and $|\varphi| > 1$, we get $|\mathscr{F}(\varphi)| < (c_1 + c_2 + 4)|\varphi|$.
- For the formula $\varphi = SCC(i, j, \psi, \phi)$, we have $|\mathscr{F}(\varphi)| = |\mathscr{F}(\psi)| + |\mathscr{F}(WCC(i, j, \psi, \phi))| + 4$. Thus, from the previous result and assumption, $\exists c_1, c_2$ s.t. $|\mathscr{F}(\varphi)| < c_1|\psi| + c_2|WCC(i, j, \psi, \phi)| + 4$. Because $|\psi| < |\varphi|$ and $|WCC(i, j, \psi, \phi)| = |\varphi|$, we obtain $|\mathscr{F}(\varphi)| < (c_1 + c_2 + 4)|\varphi|$.
- For the formula $\varphi = FuW(i, WCC(i, j, \psi, \phi))$, we have $|\mathscr{F}(\varphi)| = 2|\mathscr{F}(WCC(i, j, \psi, \phi))| + 6$. Thus, $\exists c_1$ s.t. $|\mathscr{F}(\varphi)| < 2c_1|WCC(i, j, \psi, \phi)| + 6$. Because $|WCC(i, j, \psi, \phi)| < |\varphi|$, we obtain $|\mathscr{F}(\varphi)| < (2c_1 + 6)|\varphi|$.
- For the formula $\varphi = FuS(i, SCC(i, j, \psi, \phi))$, we have $|\mathscr{F}(\varphi)| = 2|\mathscr{F}(SCC(i, j, \psi, \phi))| + |\mathscr{F}(\psi)| + 6$. Thus, $\exists c_1, c_2$ s.t. $|\mathscr{F}(\varphi)| < 2c_1|SCC(i, j, \psi, \phi)| + c_2|\psi| + 6$. So we obtain $|\mathscr{F}(\varphi)| < (2c_1 + c_2 + 6)|\varphi|$, so the proposition.

□

Proposition 3. *The time complexity of transforming the RTCTL^{cc} formula is linear with respect to the length of the input formula φ, i.e., $\mathcal{O}(|\varphi|)$.*

Proof. Algorithm 2 takes the RTCTL^{cc} formula φ as input and writes in a recursion manner the corresponding RTCTL formula according to the structure of φ. The result follows from the fact that (1) the length of the recursion is bounded by the size of the input formula φ, and (2) the size of $\mathscr{F}(\varphi)$ is bounded by the size of φ subject to a constant (from Proposition 2). □

Theorem 2. *There exists a model checking algorithm for RTCTL^{cc} formulae, which runs in time $\mathcal{O}(|M| \times |\varphi|)$.*

Proof. It is known from Emerson et al. [1992] that RTCTL model checking can be done in a linear time with respect to the size of the RTCTL model and formula, i.e., $\mathcal{O}(|\mathscr{F}(M)| \times |\mathscr{F}(\varphi)|)$. From Algorithm 1, M and $\mathscr{F}(M)$ have the same number of states and the number of transitions in $\mathscr{F}(M)$ is at most $2|T|$ where $|T|$ is the number of transitions in M. Consequently, $|\mathscr{F}(M)| \leq 2|M|$, i.e., the size of $\mathscr{F}(M)$

is linear with the size of M, and since from Proposition 3 the length of φ is linear with the length of $\mathscr{F}(\varphi)$, we conclude that the model checking of RTCTL can run in $\mathcal{O}(|M| \times |\varphi|)$. Because the transformations of the RTCTL^{cc} model and formula into the corresponding RTCTL model and formula are both linear with respect to the input size (from Propositions 2 and 3), the time complexity of the RTCTL^{cc} model checking is $\mathcal{O}(|M| \times |\varphi|) + \mathcal{O}(|M|) + \mathcal{O}(|\varphi|)$; so the result. □

Theorem 3. *The problem of RTCTL^{cc} model checking is P-complete.*

Proof. The upper bound is P, which follows from Theorem 2. The lower bound is also P, which follows directly from the P-completeness of the CTL model checking problem [Schnoebelen, 2002]. □

When M is given under the form of a synchronized product of m structures/programs/modules $M_1, M_2, \dots, M_m$ as in the symbolic model checker NuSMV, it is suitable to analyze the problem of model checking RTCTL^{cc} for concurrent modules.

Theorem 4. *The space complexity of the RTCTL^{cc} model checking problem is PSPACE-complete with respect to concurrent programs.*

Proof. In [Laroussinie et al., 2003], it is proven that model checking TCTL$_s$ (an extension of CTL with bounded until operators) is PSPACE-complete for concurrent programs and also shown that TCTL$_s$ can subsume RTCTL. Thus, the upper bound of model checking RTCTL is PSPACE. On the other hand, RTCTL can subsume CTL. Because model checking CTL is PSPACE-complete for concurrent programs [Schnoebelen, 2002], the lower bound of model checking RTCTL is PSPACE as well. We conclude that model checking RTCTL is PSPACE-complete for concurrent programs. Moreover, it is proven in [El-Kholy et al., 2014] that CTL^{cc} is PSPACE-complete for concurrent programs. Thus, the RTCTL^{cc} completeness in PSPACE follows from the PSPACE-completeness of the two fragments CTL^{cc} and RTCTL. □

Remark 1. *This result is also confirmed by the fact that (1) model checking RTCTL is PSPACE-complete for concurrent programs, and (2) the size of respectively the obtained RTCTL model and the obtained formula after transformation is linear with the size of respectively the input RTCTL^{cc} model and the input formula.*

5. Implementation

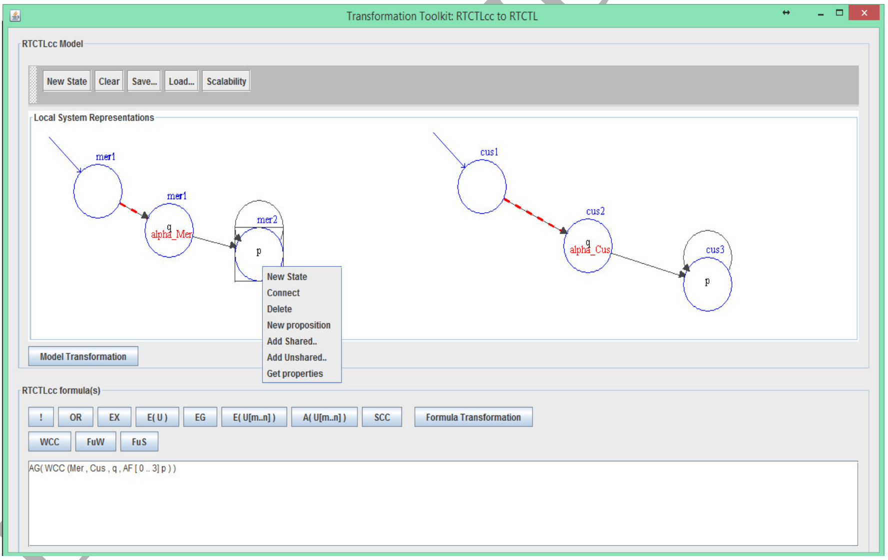
We have fully implemented a new toolkit in Java which has two main components: model transformation and formula transformation[3].

[3]The toolkit with experiments are available at: `https://users.encs.concordia.ca/~bentahar/ISPLtoNuSMVTool.jar`

5.1. Implementing and Evaluating the toolkit

The first component of the toolkit implements Algorithm 1 introduced in Section 3.1. This component is responsible for automatically transforming RTCTL^{cc} models into RTCTL models. To test and evaluate the efficiency of the toolkit regarding the model transformation, we adopted two business scenarios which are formalized using our RTCTL^{cc} model. Figure 3 shows the graphical user interface of the model transformation process. This interface enables designers to design the local system of each agent by drawing the required local states using the `New State` button in the toolbar. Each drawn state has:

1. An identifier.
2. The values of shared and/or unshared variables.
3. A set of true atomic propositions.
4. A set of allowable actions collected in a `pop-up menu` such as the `Connect` action, which links the source state with its destination state(s).



AG(WCC (Mer , Cus , q , AF [0 .. 3] p))

Figure 3: Screenshot of the graphical user interface of our toolkit

The figure specifically depicts the local system models of the customer and merchant agents constituting the MAS introduced in the following business scenario:

Scenario 1. *Let q=Payment(received) and p=Goods(delivered) be two propositions. The business scenario formalized as $\varphi = AG(WCC(Mer, Cus, q, AF^{\leq 3}p))$ specifies that along with all paths globally the merchant weakly commits to deliver goods to the customer within at most 3 days once she/he received the agreed payment.*

Notably, the dashed red arrows in the figure are the accessibility relations computed by the toolkit. When the design process is completed, the designer can automatically transform the local models of interacting agents into the SMV modules by pressing the `Model Transformation` button such that a module name is taken from an agent name. Moreover, our toolkit adds the main module to:

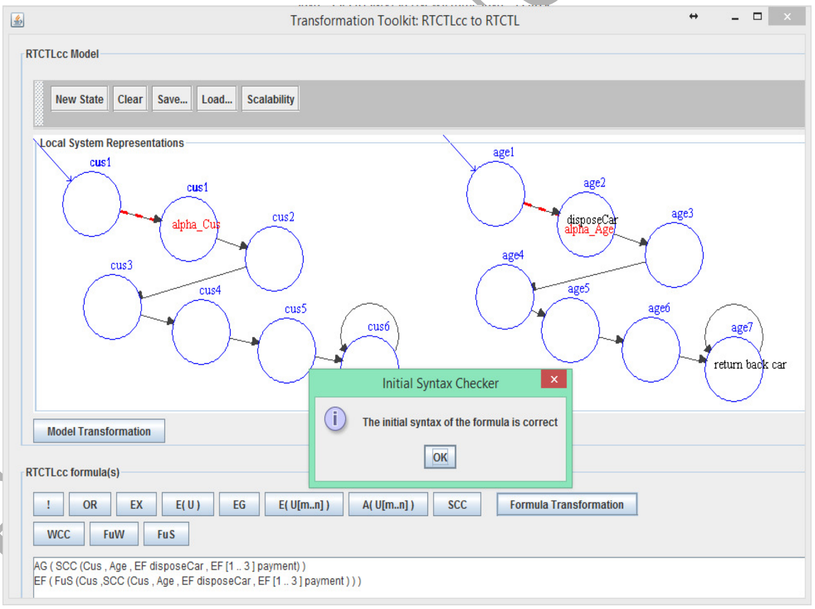
– Instantiate all SMV modules. In this scenario, the merchant and customer modules are instantiated.

– Define the required atomic propositions.

– Define formulae

– Define initialization values

The second component of our toolkit implements Algorithm 2, which is responsible for automatically transforming RTCTL^{cc} formulae into RTCTL formulae. Figure 4



The initial syntax of the formula is correct

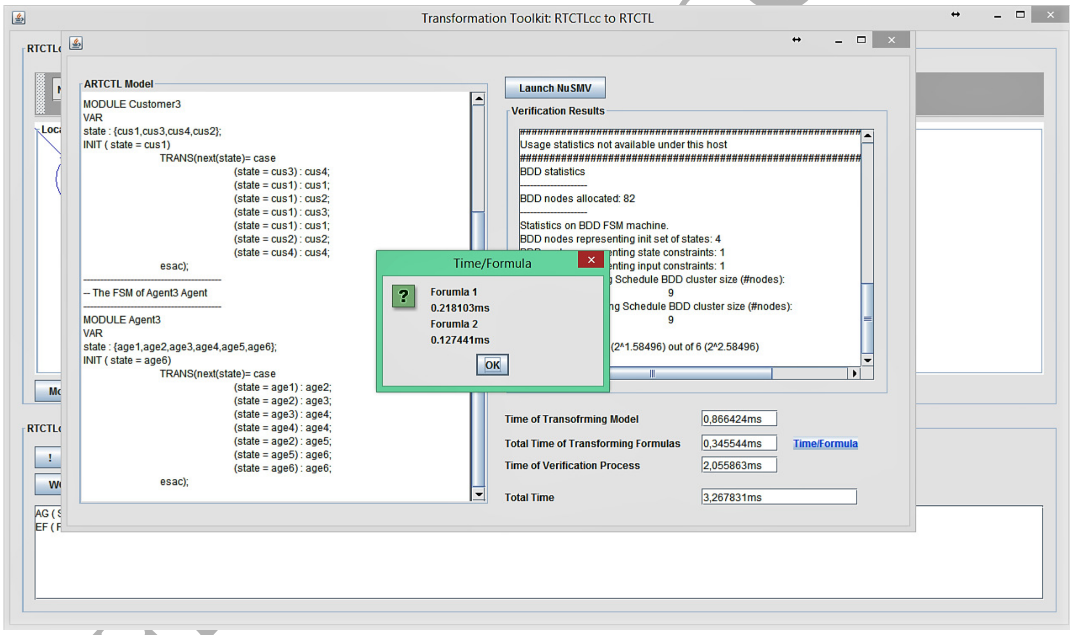
```
AG ( SCC (Cus , Age , EF disposeCar , EF [1 .. 3 ] payment) )
EF ( FuS (Cus ,SCC (Cus , Age , EF disposeCar , EF [1 .. 3 ] payment ) ) )
```

Figure 4: Screenshot of the formula transformation component and parser check

displays the RTCTL^{cc} formula panel and a set of RTCTL^{cc} operators in the form of command buttons that can help designers write, edit, and modify their desirable properties. The designers only insert the required atomic propositions. The figure also shows the quantitative formulae and the local models of the customer and agency agents in the business scenario introduced in Example 7. When all the required properties are added, the designers can use the `Formula Transformation` button to transform these properties into RTCTL formulae. Moreover, we implemented a parser that checks the legality of the RTCTL^{cc} syntax. The parser also supports other capabilities, such as displaying error messages and suggesting possible solutions to comply with the defined RTCTL^{cc} grammar. An example of a correct legal RTCTL^{cc} formula is shown in Figure 4.

5.2. *The toolkit engine*

The toolkit engine uses the NuSMV model checker as a core component to perform the verification process. Figure 5 shows, on the left, the generated NuSMV modules of the business scenario introduced in Example 7. The `Launch NuSMV` button runs



```
MODULE Customer3
VAR
state : {cus1,cus3,cus4,cus2};
INIT ( state = cus1)
            TRANS(next(state)= case
                        (state = cus3) : cus4;
                        (state = cus1) : cus1;
                        (state = cus1) : cus2;
                        (state = cus1) : cus3;
                        (state = cus1) : cus1;
                        (state = cus2) : cus2;
                        (state = cus4) : cus4;
            esac);
--------------------------------
-- The FSM of Agent3 Agent
--------------------------------
MODULE Agent3
VAR
state : {age1,age2,age3,age4,age5,age6};
INIT ( state = age6)
            TRANS(next(state)= case
                        (state = age1) : age2;
                        (state = age2) : age3;
                        (state = age3) : age4;
                        (state = age4) : age4;
                        (state = age2) : age5;
                        (state = age5) : age6;
                        (state = age6) : age6;
            esac);
```

```
##############################################################
Usage statistics not available under this host
##############################################################
BDD statistics
--------------------
BDD nodes allocated: 82
--------------------
Statistics on BDD FSM machine.
BDD nodes representing init set of states: 4
                  enting state constraints: 1
                  enting input constraints: 1
                  g Schedule BDD cluster size (#nodes):
                        9
                  ng Schedule BDD cluster size (#nodes):
                        9

                  (2^1.58496) out of 6 (2^2.58496)
```

```
Forumla 1
0.218103ms
Forumla 2
0.127441ms
```

Time of Transofrming Model	0,866424ms
Total Time of Transforming Formulas	0,345544ms
Time of Verification Process	2,055863ms
Total Time	3,267831ms

Figure 5: Screenshot of the generated NuSMV modules alongside verification results

NuSMV with a set of batched commands. Notice from the figure that the present case study has two quantitative formulae and the inserted atomic propositions are preceded with the `DEFINE` keyword, while the transformed formulae are preceded with the `SPEC` keyword in the automated main module. The verification results displayed in the right panel are the number of reachable states and total time in milliseconds. The total time (2.4994ms) is the summation of (1) the time of transforming the model, (2) the total time of transforming the formulae, and (3) the time of the verification process (or execution time). To show the transformation time of each formula, we can press on the `Time/Formula` button which pups up an information dialog box as shown in Figure 5. It is worth noticing that the time of transforming our model M, the total time of transforming the formulae and time of transforming each formula are linear with respect to the size of the inputs, which confirms our theoretical results introduced in Section 4.

5.3. Scalability aspect of the toolkit

We added another capability to our toolkit to scale MASs with respect to a certain modeling interleaved technique. In this technique, each agent is paired with another agent and all the resulting pairs move in a parallel way. Also, each desirable property is redefined in a parametric form using the conjunction operator. Our toolkit precisely starts with reading the encoding of the MAS model and generates automatically the required SMV modules according to both the given interleaved technique and the number of interacting agents. These modules are instantiated in the main module which technically includes initial conditions and atomic propositions, transformed directly from the input encoding model. The toolkit also transforms automatically parametric formulae into the corresponding ones with respect to Algorithm 2.

5.3.1. Testing and evaluating the toolkit

To test and evaluate our toolkit with respect to the scalability aspect, we used the ordering protocol introduced in [Desai et al., 2005]. The protocol specifies the rules that regulate the interaction between seller and buyer agents. It specifically begins with the buyer requesting the price quote for certain good items from the seller, which replies with an offer. The offer means creating a strong conditional commitment to deliver the requested goods when the seller accepts the buyer's price quote. The protocol ends when the buyer sends the acceptance message. The acceptance message means creating a strong conditional commitment to send the agreed payment to the seller once the buyer accepts the delivered goods.

Given the protocol specification, we embark to model check the ordering protocol. This goal is achieved by formalizing the protocol specification using our model M and by expressing a set of properties in RTCTL^{cc}. The formalized model is then encoded in the ISPL+ input language of our MCMAS+ tool introduced in [El-Kholy et al., 2014] wherein the protocol is modeled as the environment agent while participating parties are modeled as ISPL+ intelligent agents. The main motivation behind encoding the protocol model in ISPL+ is to (1) achieve the scalability aspect which requires increasing the state space of the protocol model and this increment will not be easy to consider in the graphical user interface of our toolkit, and (2) capture all aspects of our model M such as shared and unshared variables which are directly incorporated into the ISPL+ language. For the quantitative properties, we expressed the following formulae using RTCTL^{cc}:

$$\varphi_1 = EF\ SCC(Seller, Buyer, aPrice, EF^{[0..3]}\ dGoods)$$
$$\varphi_2 = EF\ SCC(Buyer, Seller, dGoods, EF^{[0..2]}\ sPayment)$$

These formulae are kind of reachability properties. The first formula checks whether or not there exists a possibility for the seller to strongly commit to delivering the requested goods to the buyer within at most three days if the buyer accepts the price. The second formula can be read in the same way. The ISPL+ encoding model is transformed into the SMV encoding module using the following transformation rules:

– Extract the set of interacting agents (seller, buyer, and environment agents) in the ISPL+ model and define the same set of SMV modules using `MODULE <Agent-Name>`.

– For each ISPL+ agent, we transform:

  – Each variable in the `Vars...end Vars` section in the ISPL+ model into the SMV variables `<v1>, ..., <vn>` with the same data types using the `VAR` statement. These variables include local states and shared and unshared variables.

  – Local actions in the `Actions` section in the ISPL+ model into input variables in the SMV module using the `IVAR` statement.

  – The local transitions between local states into the `TTRANS` statement. In this statement, we transform the initial conditions of each ISPL+ agent into the initial conditions using the `TINIT` expression and use the `next` and `case` expressions to define transitions wherein the labeled actions are used as guards to fire these transition.

– The defined SMV modules are instantiated in the main module using the `VAR` statement. In the main module, we also:

  – Transform the defined atomic propositions in the ISPL+ program into the set of atomic propositions which are defined by the `define` statement.

  – Define the needed atomic propositions in the accessible states using the `define` statement.

The above transformation process is repeated n times where n is the number of agents and the generated SMV modules should be paired according to our interleaved technique. To complete the transformation process, we used Algorithm 2 to transform the above RTCTL^{cc} formulae (φ_1 and φ_2) into RTCTL formulae where each formula is preceded by the keyword `SPEC`.

Table 3 reports 12 experiments of verifying the compliance of the ordering protocol with the transformed formulae of φ_1 and φ_2, namely φ'_1 and φ'_2 using Algorithm 2. Our experiments were performed on a Laptop with the following specification: (1) processor is Intel(R) Core(TM) i7-7820HQ CPU, 4 cores and 8 MB Cache at 2.9 GHz, (2) installed memory (RAM) is 16 GB (DDR3), and (3) operating system is 64-bit operating system, Windows 8.1. The transformed formulae φ'_1 and φ'_2 are redefined

Table 3: Verification results of 12 experiments

Exp. #	**no. of agents**	**no. of reachable states**	**time of transf. & param. φ_1, φ_2 in ms**	**time of transf. model in ms**	**avera. total time in ms**
1	3	5	0.118132, 0.113656	07.00690	9.238688
2	6	25	0.118759, 0.117504	10.414627	12.75089
3	9	125	0.124811, 0.122373	12.253960	14.80114
4	12	625	0.131261, 0.133563	17.273747	19.93857
5	15	3125	0.142771, 0.149051	18.381369	21.27313
6	18	15625	0.156720, 0.159151	20.361401	23.67727
7	21	78125	0.167575, 0.164546	20.997321	24.92944
8	24	390625	0.171424, 0.176421	26.318603	30.96645
9	30	9.76563e+06	0.190733, 0.192444	29.235256	34.61843
10	36	2.44141e+08	0.2296500, 0.229650	30.089709	36.64260
11	42	6.10352e+09	0.260013, 0.269451	31.693837	39.22330
12	66	2.38419e+15	0.449892, 0.450746	51.885031	117.0266699

in a parametric form through our experiments. For example, the parametric form of φ'_1 in Experiment 7 is generated with the conjunction operator as follows:

$$\varphi'_1 = \bigwedge_{i=1}^{n} EF\big[EX(aPrice \wedge \alpha^{seller_i}\alpha^{buyer_i}) \wedge AX\big((aPrice \wedge \alpha^{seller_i}\alpha^{buyer_i}) \Rightarrow EF^{[0..3]}\, dGoods\big)\big]$$

where the number of agents is 21 (i.e., seven seller agents, seven buyer agents, and seven environment agents). Analyzing the reported verification results in Table 3 reveals that the number of reachable states reflecting the state space increases exponentially when the number of agents increases. The transformation times of the models and formulae in milliseconds increase polynomially. The increase in the average total time (where the total time is the summation of the total time of transforming formulae, time of transforming model, and time of verification process) is also polynomial. These experimental results confirm the theoretical ones.

5.4. Real and industrial case study

Our real and industrial case study is called aircraft landing gear system (LGS). This system supports the airplane during the landing, taking off, and taxiing without human loss and critical damages. Therefore, this system is one of the most critical subsystems of an aircraft. Boniol and Wiels recently proposed a full description of the landing gear system in [Boniol and Wiels, 2014]. We summarize and classify their description as follows:

1. The elementary elements
   The basic elements of the LGS system are gears, doors, and software.
   – LGS has specifically three landing sets located in the left, front, and right side of the aircraft. Each one of the landing sets includes a gear which can be extended, retracted, or maneuvered. Moreover, each landing set includes a door which can be closed, open, or maneuvered. The LGS system is governed by a customized software and can be in two modes: normal or emergency.
   – The emergency mode can be detected by the software. In the emergency mode, the system stops and the mechanical parts start working mechanically.
2. The pilot and its main functions
   – The pilot interface has a handle switch with two directional positions: DOWN and UP. If the gears are retracted and the handle switch is going from UP to DOWN, the extending process is carried out as follows: open doors, extend gears, and close doors.
   – If the gears are extended and the handle switch is going from DOWN to UP, the retracting process is carried out as follows: open doors, retract gears, and close doors.
   – At any instant time during the door/gear maneuver, this maneuver can be terminated and reversed by the pilot via moving the handle switch in the other direction.
   – The pilot interface has three lights in the cockpit: green, orange, and red. If the gears are extended, the green light is ON and if the gears are maneuvering, the orange light is ON. In the emergency mode, the red light is ON.

3. Mechanical parts
   - There are five electro-valves working as follows wherein each electro-valve can be ON or OFF: one general electro-valve that provides or removes pressure in the hydraulic circuit and two specific electro-valves that open or close doors and two specific electro-valves that retract or extend gears.
   - If the general electro-valve is ON, the pressure will be eventually provided in the hydraulic circuit and when it is OFF, the pressure will be removed from the hydraulic circuit.
   - If the status of the handle is changed, the general electro-valve is set to ON if it is OFF. When a gear operation is complete, the general electro-valve is set to OFF.
   - Any specific electro-valve can be set to ON, only when the pressure in the hydraulic circuit is provided.
   - If the door is closing or opening and the corresponding electro-valves are ON, the doors will be eventually closed or open. If the gear is retracting or extending and the corresponding electro-valves are stimulated, the gears will be eventually retracted or extended.
4. Analogical switch
   - There is an analogical switch which is in charge of making the connection between the software and the general electro-valve. The analogical switch can be closed or open.
   - If the analogical switch is not already closed, it will be mechanically closed each time there is a change in the pilot's handle. 40 seconds after the last handle change, the switch is mechanically turned open. Only when the switch is closed, the software can send information successfully to the general electro-valve.
5. Software inputs and outputs
   - The software receives the following inputs: (1) one input from the handle, one input from the analogical switch, and one input from the circuit pressurized, and (2) three inputs from the gears extension and three inputs from the gears retraction. Additionally, three inputs from the doors closed and three inputs from the doors opened.
   - The software sends the following outputs: (1) one output to the general electro-valve, one output to the closed door electro-valve, and one output to the open door electro-valve, and (2) one output to the retraction electro-valve and one output to the extension electro-valve.

– The software sends the following outputs to the cockpit: when gears are maneuvering, the orange light is so ON, when gears are locked down, the green light is so ON, and when anomaly happens, the red light is so ON.

Now, we use our model $M = (S, I, T, \{\sim_{i\to j} \mid (i, j) \in \mathcal{A}^2\}, \mathcal{V})$ to formalize the aircraft landing system where $\mathcal{A} = \{Pilot, Software, Emergency\}$. Notice that the behaviors of gears and doors are included in the software agent and the emergency agent as they are shared between them. We then proceed to encode the pilot, software, and emergency agents in the ISPL+ input language of MCMAS+. Specifically, we encode local states, shared and unshared variables, local protocol/policy, local actions, local transitions, and initial states for each agent. We validate our modeling using the capability in the MCMAS+ tool called `Explicit Interactive Mode` to have the model as it is intended.

In addition to the description of the landing gear system, Boniol and Wiels introduced a set of requirements in English language [Boniol and Wiels, 2014]. For example,

> "(R11) When the command line is working (normal mode), if the landing gear command handle has been pushed DOWN and stays DOWN, then the gears will be locked down and the doors will be seen closed less than 15 seconds after the handle has been pushed."

The R11 requirement can be expressed in RTCTL as follows:

$$AG(AG(\mathit{PressedDown}) \Rightarrow AF^{\leq 15}(\mathit{GearsLocked} \wedge \mathit{DoorsClosed}))$$

While the events occurrence and the receiving responses are timely constrained, it is not clear in this formalization (1) who performs these events (e.g., *PressedDown* and *GearsLocked*), (2) what is the ordering of these events, (3) is there any relation between the components that could perform these events, and (4) what is the nature of this relation. This relation exists in the real case study and is performed mechanically through communication among involved parts. All these questions can be addressed if we use our RTCTL^{cc} to formalize R11 as follows:

$$AG\ WCC((\mathit{Software}, \mathit{Pilot}, AG(\mathit{PressedDown}), AF^{\leq 15}(\mathit{GearsLocked} \wedge \mathit{DoorsClosed})))$$

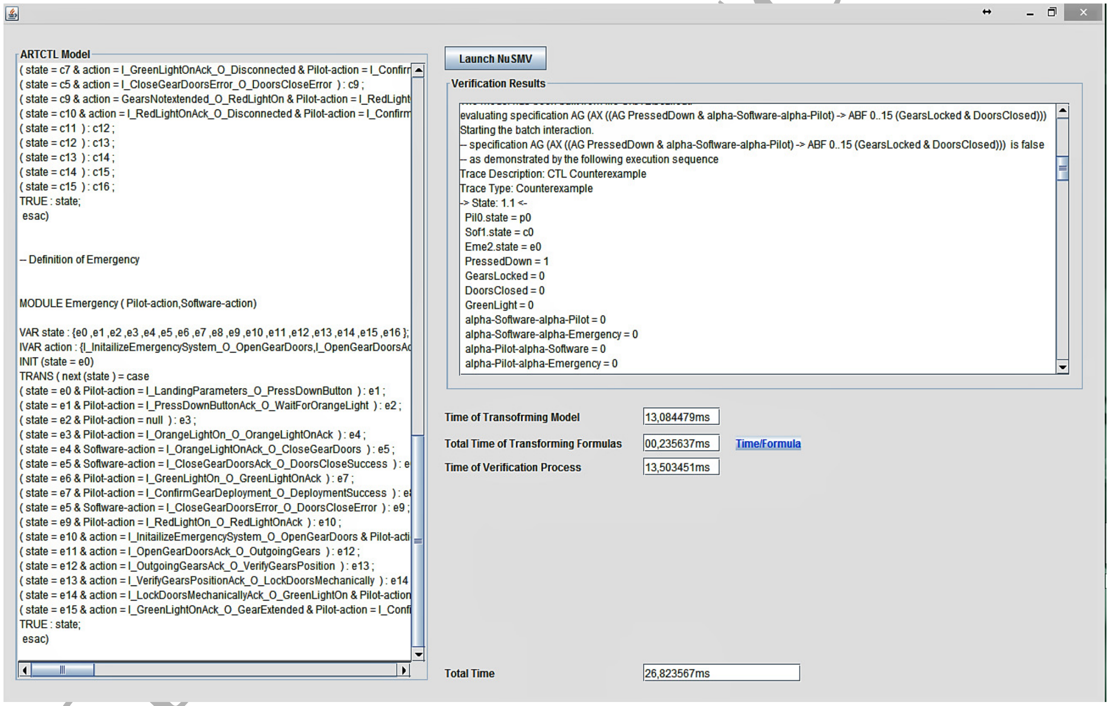
This formula means that along all paths in all states the software agent commits to lock gears and close doors in no more than 15 seconds if the pilot agent changes the handle position from up to down. The communication and direct relation between interacting agents are modeled using conditional commitments. Such commitments are obligatory contracts. Moreover, we express the safety and liveness properties in RTCTL^{cc} as follows:

$$AG\neg(\mathit{PressedDown} \wedge AG(\neg \mathit{GreenLight}))$$

$$EF(RedLight \wedge EF\ \ GreenLight)$$

Here, the bad situation in the safety property means that the pilot agent pressed the handle to down but there is no possibility to lit the green light. The liveness property means that there is a path in its future when the red light is lit, there is a possibility to lit the green light by the emergency agent.

By so doing, we used our transformation tool to transform the ISPL+ model and formulae into the SMV model to be able to start the verification process using NuSMV. However, the NuSMV tool reported a counterexample showing why the formula $AG\ \ WCC((Software, Pilot, AG(PressedDown), AF^{\leq 15}\ (GearsLocked \wedge DoorsClosed)))$ is false[4] (see Figure 6). We used this counterexample to correct our ISPL+ model and re-run the transformation tool to get the good SMV model.



ARTCTL Model

```
( state = c7 & action = I_GreenLightOnAck_O_Disconnected & Pilot-action = I_Confirr
( state = c5 & action = I_CloseGearDoorsError_O_DoorsCloseError  ) : c9 ;
( state = c9 & action = GearsNotextended_O_RedLightOn & Pilot-action = I_RedLight
( state = c10 & action = I_RedLightOnAck_O_Disconnected & Pilot-action = I_Confirm
( state = c11  ) : c12 ;
( state = c12  ) : c13 ;
( state = c13  ) : c14 ;
( state = c14  ) : c15 ;
( state = c15  ) : c16 ;
TRUE : state;
 esac)


-- Definition of Emergency


MODULE Emergency ( Pilot-action,Software-action)

VAR state : {e0 ,e1 ,e2 ,e3 ,e4 ,e5 ,e6 ,e7 ,e8 ,e9 ,e10 ,e11 ,e12 ,e13 ,e14 ,e15 ,e16 };
IVAR action : {I_InitailizeEmergencySystem_O_OpenGearDoors,I_OpenGearDoorsAc
INIT (state = e0)
TRANS ( next (state ) = case
( state = e0 & Pilot-action = I_LandingParameters_O_PressDownButton  ) : e1 ;
( state = e1 & Pilot-action = I_PressDownButtonAck_O_WaitForOrangeLight  ) : e2 ;
( state = e2 & Pilot-action = null  ) : e3 ;
( state = e3 & Pilot-action = I_OrangeLightOn_O_OrangeLightOnAck  ) : e4 ;
( state = e4 & Software-action = I_OrangeLightOnAck_O_CloseGearDoors  ) : e5 ;
( state = e5 & Software-action = I_CloseGearDoorsAck_O_DoorsCloseSuccess  ) : e
( state = e6 & Pilot-action = I_GreenLightOn_O_GreenLightOnAck  ) : e7 ;
( state = e7 & Pilot-action = I_ConfirmGearDeployment_O_DeploymentSuccess  ) : e
( state = e5 & Software-action = I_CloseGearDoorsError_O_DoorsCloseError  ) : e9 ;
( state = e9 & Pilot-action = I_RedLightOn_O_RedLightOnAck  ) : e10 ;
( state = e10 & action = I_InitailizeEmergencySystem_O_OpenGearDoors & Pilot-acti
( state = e11 & action = I_OpenGearDoorsAck_O_OutgoingGears  ) : e12 ;
( state = e12 & action = I_OutgoingGearsAck_O_VerifyGearsPosition  ) : e13 ;
( state = e13 & action = I_VerifyGearsPositionAck_O_LockDoorsMechanically  ) : e14
( state = e14 & action = I_LockDoorsMechanicallyAck_O_GreenLightOn & Pilot-action
( state = e15 & action = I_GreenLightOnAck_O_GearExtended & Pilot-action = I_Confi
TRUE : state;
 esac)
```

Verification Results

```
evaluating specification AG (AX ((AG PressedDown & alpha-Software-alpha-Pilot) -> ABF 0..15 (GearsLocked & DoorsClosed)))
Starting the batch interaction.
-- specification AG (AX ((AG PressedDown & alpha-Software-alpha-Pilot) -> ABF 0..15 (GearsLocked & DoorsClosed)))  is false
-- as demonstrated by the following execution sequence
Trace Description: CTL Counterexample
Trace Type: Counterexample
-> State: 1.1 <-
 Pil0.state = p0
 Sof1.state = c0
 Eme2.state = e0
 PressedDown = 1
 GearsLocked = 0
 DoorsClosed = 0
 GreenLight = 0
 alpha-Software-alpha-Pilot = 0
 alpha-Software-alpha-Emergency = 0
 alpha-Pilot-alpha-Software = 0
 alpha-Pilot-alpha-Emergency = 0
```

Time of Transofrming Model: 13,084479ms

Total Time of Transforming Formulas: 00,235637ms

Time of Verification Process: 13,503451ms

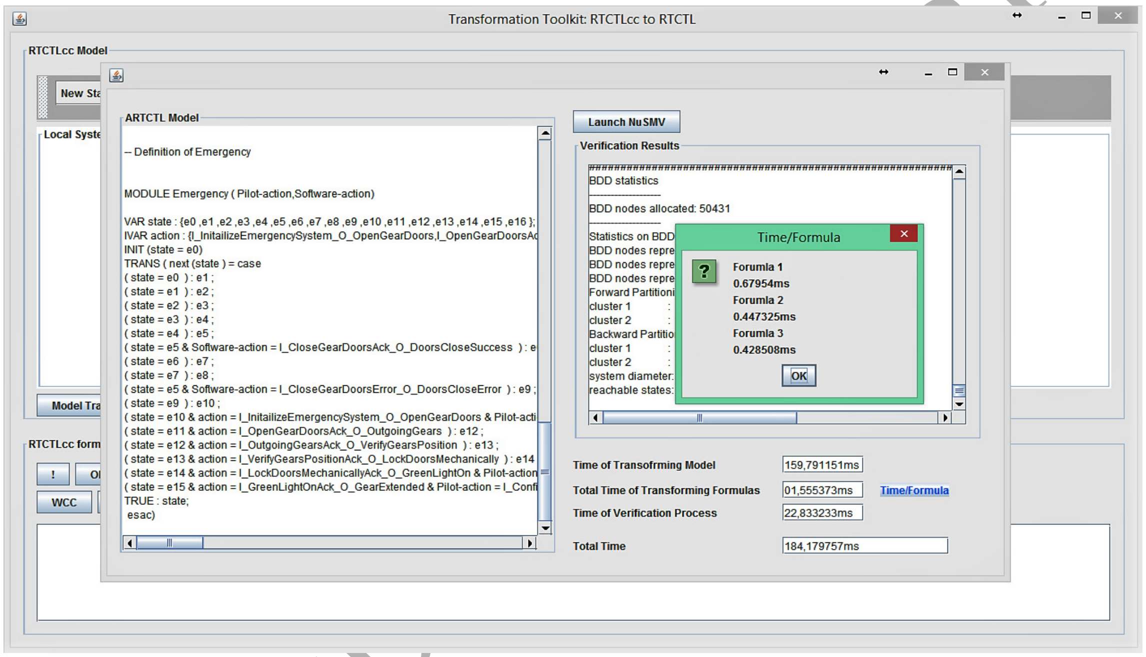
Total Time: 26,823567ms

Figure 6: A counterexample explaining why the formula $AG\ WCC((Software, Pilot, AG(PressedDown), AF^{\leq 15}\ (GearsLocked \wedge DoorsClosed)))$ is false

Moreover, we verified again all tested formulae against the corrected SMV model

[4]The ISPL+ code of this experiment is available in the sub-folder "CaseStudy" of the folder "experiments" available in the toolkit at: `https://users.encs.concordia.ca/~bentahar/ISPLtoNuSMVTool.jar`

and illustrated their evaluation in Figure 7. As shown in the figure, all the formulae hold in the second run. The transformed ISPL+ landing gear system model into the NuSMV model, the verification results, the time of transforming ISPL+ model, the total time of transforming RTCTL^{cc} formulae, the time of verification process, the total time, and the transformation time of each formula are also illustrated in Figure 7.



```
-- Definition of Emergency

MODULE Emergency ( Pilot-action,Software-action)

VAR state : {e0 ,e1 ,e2 ,e3 ,e4 ,e5 ,e6 ,e7 ,e8 ,e9 ,e10 ,e11 ,e12 ,e13 ,e14 ,e15 ,e16 };
IVAR action : {I_InitailizeEmergencySystem_O_OpenGearDoors,I_OpenGearDoorsAc
INIT (state = e0)
TRANS ( next (state ) = case
( state = e0  ) : e1 ;
( state = e1  ) : e2 ;
( state = e2  ) : e3 ;
( state = e3  ) : e4 ;
( state = e4  ) : e5 ;
( state = e5 & Software-action = I_CloseGearDoorsAck_O_DoorsCloseSuccess  ) : e
( state = e6  ) : e7 ;
( state = e7  ) : e8 ;
( state = e5 & Software-action = I_CloseGearDoorsError_O_DoorsCloseError  ) : e9 ;
( state = e9  ) : e10 ;
( state = e10 & action = I_InitailizeEmergencySystem_O_OpenGearDoors & Pilot-acti
( state = e11 & action = I_OpenGearDoorsAck_O_OutgoingGears  ) : e12 ;
( state = e12 & action = I_OutgoingGearsAck_O_VerifyGearsPosition  ) : e13 ;
( state = e13 & action = I_VerifyGearsPositionAck_O_LockDoorsMechanically  ) : e14
( state = e14 & action = I_LockDoorsMechanicallyAck_O_GreenLightOn & Pilot-action
( state = e15 & action = I_GreenLightOnAck_O_GearExtended & Pilot-action = I_Confi
TRUE : state;
 esac)
```

```
################################################################
BDD statistics
--------------------
BDD nodes allocated: 50431
--------------------
Statistics on BDD
BDD nodes repre
BDD nodes repre
BDD nodes repre
Forward Partitioni
cluster 1         :
cluster 2         :
Backward Partitio
cluster 1         :
cluster 2         :
system diameter:
reachable states:
```

Forumla 1
0.67954ms
Forumla 2
0.447325ms
Forumla 3
0.428508ms

Time of Transofrming Model 159,791151ms
Total Time of Transforming Formulas 01,555373ms
Time of Verification Process 22,833233ms
Total Time 184,179757ms

Figure 7: Screenshot of the generated SMV model (on the left) alongside the verification results, the transformation times, and the total time (on the right)

6. Related work

6.1. *Formal transformation techniques*

The approaches that we discuss hereafter should have a formal semantics for social commitments to be able to (1) implement model checking, and (2) define systematic rules to transform their semantic definitions. To apply this constraint, commitments should be modeled as temporal modalities. Therefore, there are merely two formal approaches in the literature that have defined semantics for conditional commitments, which cope with conditional business contracts. The first approach is the one presented by Singh [2008]. The author specifically extended LTL with two temporal modalities to represent and reason about two types of conditional commitments (called practical and dialectical commitments). The authors in [Baldoni et al., 2015] built on Singh's dialectical commitments and introduced an approach to compose multi-agent protocols and verify the compliance of the participating agents with the social states. In this approach, the semantics of dialectical commitments (or claims) are defined using the precedence logic. This logic has three primary operators: choice, concurrence, and before. Their verification process is theorem proving-like.

In the second approach, we extended CTL with four modalities to represent and reason about two types of conditional commitments (weak and strong) and their fulfillment [El-Kholy et al., 2014]. The resulting logical language is so-called CTL^{cc}. The model checking problem is addressed by developing symbolic algorithms implemented on top of MCMAS. Moreover, the proposed technique tackles the open problem of automatically verifying the Singh's semantic models. This is because we showed in [El-Kholy et al., 2014] that the semantics of weak and strong commitments achieve all the reasoning rules introduced by Singh [2008].

Because unconditional commitments $C(i, j, \varphi)$ can be treated as a special case of conditional commitments when the antecedent is true, i.e., $C(i, j, \varphi) \equiv WCC(i, j, \top, \varphi)$ $\equiv SCC(i, j, \top, \varphi)$, we then continue this subsection by discussing the current transformation techniques that have been developed to transform the problem of model checking unconditional commitment logics. Among these approaches, El-Menshawy et al. [2011] formally transformed the problem of model checking CTLC (an extension of CTL with unconditional commitment modality) into the problems of model checking CTLK (an extension of CTL with knowledge modality) and ARCTL (an extension of CTL with action formulae), so that the use of the MCMAS and extended NuSMV model checkers is made possible.

El-Menshawy et al. [2013] improved the definition of the accessibility relation introduced in [El-Menshawy et al., 2011] to have a new semantics for unconditional commitment and fulfillment modalities. The new logic is called CTLC$^+$. Then, they formally transformed the problem of model checking CTLC$^+$ into the problems of model checking ARCTL and GCTL* (a generalized version of CTL* with action formulae). This transformation technique enabled them to use the extended NuSMV symbolic model checker and the CWB-NC automata-based model checker as a benchmark.

The authors in [El-Menshawy et al., 2013] developed a branching time temporal logic called ACTL^{*c} by extending CTL* with temporal modalities to represent and reason about unconditional commitments and all related actions. They formally transformed the problem of model checking ACTL^{*c} into the problem of model checking GCTL* to make CWB-NC usable.

The authors in [Al-Saqqar et al., 2015] introduced a temporal logic called CTLKC$^+$, a combination of CTL modalities, knowledge modality and unconditional commitment modality. The problem of model checking CTLKC$^+$ has been formally transformed into the problem of model checking ARCTL and the extended version of NuSMV has been utilized.

Mallya et al. [2004] enriched CTL with (1) predicates to reason about commitments and fulfillment and violation actions, and (2) two existential and universal quantifiers to capture temporal deadlines in the unconditional commitment consequences without considering the verification problem. It has been shown in [El-Kholy et al., 2015], that the interval bound until operators alongside existential and universal quantifiers on paths can model Mallya et al.'s temporal quantifiers in a reasonable way. However, our quantified time intervals are not abstracted as propositions, as done in [Mallya et al., 2004]. Moreover, the model checking problem of the resulting Mallya et al.'s logic is still an open problem.

On the one hand, it is known that temporal logics are time-abstract with regard to the occurrence of events in the past and future without referencing the precise timing of events. Therefore, temporal logic-based approaches discussed above are not suitable to represent and reason about deadlines of commitments that incorporate metrics or real-time constraints as in real-life business scenarios, as argued in [Chesani et al., 2013]. On the other hand, except [Al-Saqqar et al., 2015], it is not fully discussed how the transformation tools in the literature are implemented. The authors in [Al-Saqqar et al., 2015] implemented their tool with a graphical user interface in the form of questions that are asked to the designers in order to build the CTLKC$^+$ models. The target of this transformation tool is a labeled transition system, like all the discussed approaches where transitions are labeled with actions. However, in our proposal, Kripke structures are generated where only states are labeled, which reduces the size of the transformed models, thanks to the fact of avoiding additional transitions. Because our logic and target model are entirely different, we developed and implemented a new transformation algorithm along with another algorithm to automatically compute the accessibility relations and accessible states, which would help designers design, transform and verify RTCTL^{cc} models in an easy way. Moreover, we developed a parser, visual local system representations of agents and an automatic way to scale MASs, which are missing in [Al-Saqqar et al., 2015]. For example, missing the automatic capability of reading the input model in [Al-Saqqar et al., 2015] makes the transformation technique arduous and time-consuming because it is based on asking the user for each component of each agent a set of predefined questions (thinking of how many questions are needed to transform several agents). By comparing the current approach with the discussed approaches that use transformation techniques in terms of the number of agents, we found that our approach supports more agents, 66 agents, compared to only four agents in [El-Menshawy et al., 2011], 6 agents in [El-Menshawy et al., 2013], 8 agents in [El-Menshawy et al., 2013], and 9 agents in [Al-Saqqar et al., 2015], thanks to the reduced size of the obtained model in our approach. Furthermore, all the discussed approaches neglected to investigate the computational complexity of the transformation algorithms.

6.2. Modeling deadlines of commitments

The literature of agent communication covers run-time and design-time verification techniques. Unlike our design-time technique, in run-time verification techniques, unconditional commitments are modeled as fluents in executable action languages [Chesani et al., 2013; Desai and Singh, 2007; Yolum and Singh, 2004] such as event calculus and causal logic C+. A fluent is a property, which can have different values at different time points or can hold within time intervals. The current approaches use Boolean fluents, which have two possible values: true (hence commitments hold) and false (hence commitments do not hold). The operational semantics of commitment actions is defined by a set of axioms. In the event calculus formalism, this operational semantics is as follows: action occurrences are defined by the use of *happens* predicate, the effects of actions are defined by the use of *initiates* and *terminates* predicates and the fluents values are defined by the use of *initially, holdsAt* and *holdsFor* predicates. Although these executable action languages are very easily and efficiently implemented for executable system specifications, there is no formal semantics for unconditional commitments that can be model checked.

Chesani et al. [2013] extended the event calculus formalism with data, variables and metric time to deal with temporal aspects (e.g., deadlines). The authors then used event calculus axioms to define operational semantics of unconditional commitments and their actions where commitments are modeled as fluents. We showed in [El-Kholy et al., 2015] that these axioms can be defined using the RTCTL^{cc} logic. However, because Chesani et al.'s formalism is, in fact, a first-order logic, it is then undecidable, i.e., we cannot write a program that can work for all kinds of formulae. Moreover, Chesani et al. stated that it is hard to verify commitment protocol properties (e.g., safety and liveness) which are typically checked using model checking techniques.

Chopra and Singh [2015] recently proposed a first-order language called Cupid to specify conditional commitments with respect to the information-centric aspect. This language is able to systematically treat commitment instances and supports features such as deadlines, nested conditional commitments, and complex event expressions. It can also be mapped into relational database queries to retrieve commitment instances (violated and discharged). However, this language suffers from the same limitations as Chesani et al.'s formalism [Chesani et al., 2013]. Other contributed approaches that model deadlines of unconditional commitments are evaluated and reviewed in [Chopra and Singh, 2015]. To this end, we argue that our approach can complement run-time verification techniques in which the designers can begin with model checking to verify the compliance of MASs with desirable properties expressed in our logic before starting to track the behaviors of agents at run-time. The idea is to ensure that the observed bad behaviors do not result from errors in the design specifications.

7. Conclusion and future work

The main contributions of the present article are to analyze the computational complexity of the problem of model checking RTCTL^{cc} and to develop two transformation algorithms for addressing this problem. The analyzed time and space complexities of this problem are P-complete and PSPACE-complete for explicit models and concurrent programs. These results cope with the corresponding ones of the problem of model checking CTL. Our algorithms are implemented in a new toolkit to automatically transform the problem of model checking RTCTL^{cc} into the problem of model checking RTCTL. This toolkit is implemented on top of the NuSMV model checker. The soundness and completeness of our transformation technique are proved. Moreover, the validity and usability of the toolkit are demonstrated through a set of business scenarios. The toolkit also has a capability to automatically scale MASs using the modeling interleaved technique and automatically verify their correctness against transformed parametric formulae. The scalability aspect is evaluated using the standard ordering protocol with 66 agents having a large state-space (approximately 2.38419e+15 states). It is not only the first time to study this number of interacting agents and reachable states, but the work also provides an efficient way of dealing with the timing requirements and behaviors of MASs. Finally, the feasibility of our toolkit is successfully checked using the critical aircraft landing gear system, which gives further validation to the proposed approach.

As future work, we plan to consider other conditional commitment actions such as withdraw, release, and delegate. We also plan to develop symbolic algorithms for bounded operators and other actions and then implement them on top of our symbolic model checker MCMAS+ introduced in [El-Kholy et al., 2014] in order to compare between direct and indirect verification techniques. We subsequently plan to consider arbitrary transitions in our logical model to reduce extra verification work resulting from the use of unit transition steps.

References

F. Chesani, P. Mello, M. Montali, P. Torroni, Representing and monitoring social commitments using the event calculus, Autonomous Agents and Multi-Agent Systems 27 (1) (2013) 85–130.

A. K. Chopra, M. P. Singh, Cupid: Commitments in relational algebra, in: B. Bonet, S. Koenig (Eds.), Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, AAAI Press, 2052–2059, 2015.

M. El-Menshawy, J. Bentahar, H. Qu, R. Dssouli, On the verification of social commitments and time, in: L. Sonenberg, P. Stone, K. Tumer, P. Yolum (Eds.), AAMAS, IFAAMAS, 483–490, 2011.

M. El-Menshawy, J. Bentahar, W. El-Kholy, P. Yolum, R. Dssouli, Computational logics and verification techniques of multi-agent commitments: Survey, The Knowledge Engineering Review- Cambridge 30 (5) (2015) 564–606.

W. El-Kholy, J. Bentahar, M. El-Menshawy, H. Qu, R. Dssouli, Conditional commitments: Reasoning and model checking, ACM Transaction on Software Engineering and Methodology 24 (2014) 9:1–9–49.

P. Torroni, F. Chesani, P. Mello, M. Montali, Social commitments in time: Satisfied or compensated, in: M. Baldoni, J. Bentahar, M. B. van Riemsdijk, J. Lloyd (Eds.), DALT, vol. 5948 of *LNCS*, Springer, 228–243, 2010.

Y. Moy, E. Ledinot, H. Delseny, V. Wiels, B. Monate, Testing or formal verification: DO-178C alternatives and industrial experience, IEEE Software 30 (3) (2013) 50–57.

E. Clarke, O. Grumberg, D. Peled, Model checking, MIT Press, 1999.

P. Schnoebelen, The complexity of temporal logic model checking, in: Advances in Modal Logic, vol. 4, 1–44, 2002.

R. Koymans, Specifying real-time properties with metric temporal logic, Real-time Systems 2 (1990) 255–299.

J. Ouaknine, J. Worrell, Some Recent Results in Metric Temporal Logic, in: F. Cassez, C. Jard (Eds.), Proceedings of the 6th International conference on Formal Modeling and Analysis of Timed Systems, vol. 5215, Springer, 1–13, 2008.

R. Alur, T. Feder, T. A. Henzinger, The Benefits of Relaxing Punctuality, J. ACM 43 (1) (1996) 116–146.

W. El-Kholy, M. El-Menshawy, J. Bentahar, H. Qu, R. Dssouli, Formal specification and automatic verification of conditional commitments, IEEE Intelligent Systems 30 (2015) 36–44.

R. Fagin, J. Halpern, Y. Moses, M. Vardi, Reasoning about knowledge, MIT Press, 1995.

F. Boniol, V. Wiels, The Landing Gear System Case Study, in: F. Boniol, VirginieWiels, Y. A. Ameur, K.-D. Schewe (Eds.), ABZ 2014 Case Study Track, vol. 433, Springer, 1–18, 2014.

W. El-Kholy, M. El-Menshawy, A. Laarej, J. Bentahar, F. Al-Saqqar, R. Dssouli, Real-time conditional commitment logic, in: Q. Chen, P. Torroni, S. Villata, J. Y. Hsu, A. Omicini (Eds.), PRIMA 2015: Principles and Practice of Multi-Agent Systems, vol. 9387 of *LNCS*, Springer, 547–556, 2015.

I. García-Magariño, G. P. Navarro, A model-driven approach for constructing ambient assisted-living multi-agent systems customized for Parkinson patients, Journal of Systems and Software 111 (2016) 34–48.

E. A. Emerson, A. K. Mok, A. P. Sistla, J. Srinivasan, Quantitative temporal reasoning, Real-Time Systems 4 (4) (1992) 331–352.

A. Cimatti, E. M. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, A. Tacchella, NuSMV 2: An open source tool for symbolic model checking, in: E. Brinksma, K. G. Larsen (Eds.), Proceedings of the 14th International Conference on CAV, vol. 2404 of *LNCS*, Springer, 359–364, 2002.

G. R. Santhanam, Qualitative optimization in software engineering: A short survey, Journal of Systems and Software 111 (2016) 149–156.

E. M. Clarke, E. A. Emerson, J. Sifakis, Model checking: algorithmic verification and debugging, Communications of the ACM 52 (11) (2009) 74–84.

J. Bentahar, M. El-Menshawy, H. Qu, R. Dssoulia, Communicative commitments: Model checking and complexity analysis, Knowledge-Based Systems 35 (2012) 21–34.

M. El-Menshawy, J. Bentahar, W. El-Kholy, R. Dssouli, Reducing model checking commitments for agent communication to model checking ARCTL and GCTL*, Autonomous Agent Multi-Agent Systems 27 (3) (2013) 375–418.

A. Lomuscio, C. Pecheur, F. Raimondi, Automatic Verification of Knowledge and Time with NuSMV, in: IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007, AAAI Press, 1384–1389, 2007.

M. El-Menshawy, J. Bentahar, R. Dssouli, Symbolic model checking commitment protocols using reduction, in: A. Omicini, S. Sardina, W. Vasconcelos (Eds.), DALT, vol. 6619 of *LNAI*, Springer, 185–203, 2011.

F. Laroussinie, P. Schnoebelen, M. Turuani, On the expressivity and complexity of quantitative branching-time temporal logics, Theoretical Computer Science 297 (2003) 297–315.

N. Desai, A. Mallya, A. Chopra, M. Singh, Interaction protocols as design abstractions for business processes, IEEE Transactions on Software Engineering 31 (12) (2005) 1015–1027.

M. Singh, Semantical considerations on dialectical and practical commitments, in: Proceedings of the 23th AAAI Conference on Artificial Intelligence, 176–181, 2008.

M. Baldoni, C. Baroglio, A. K. Chopra, M. P. Singh, Composing and verifying commitment-based multiagent protocols, in: Q. Yang, M. Wooldridge (Eds.), Proceedings of the 24th International Joint Conference on Artificial Intelligence IJCAI, AAAI Press, 10–17, 2015.

M. El-Menshawy, J. Bentahar, W. El-Kholy, R. Dssouli, Verifying conformance of multi-agent commitment-based protocols, Expert Systems with Applications 40 (1) (2013) 122–138.

F. Al-Saqqar, J. Bentahar, K. Sultan, W. Wan, E. Khosrowshahi, Model checking temporal knowledge and commitments in multi-agent systems using reduction, Simulation Modelling Practice and Theory 51 (2015) 45–68.

A. U. Mallya, P. Yolum, M. P. Singh, Resolving commitments among autonomous agents, in: F. Dignum (Ed.), ACL, vol. 2922 of *LNCS*, Springer, 166–182, 2004.

N. Desai, M. Singh, A modular action description language for protocol composition, in: Proceedings of the 22nd AAAI Conference on Artificial Intelligence, 962–967, 2007.

P. Yolum, M. P. Singh, Reasoning about commitments in the event calculus: An approach for sepcifying and executing protocols, Annals of Mathematics and Artificial Intelligence 42 (1–3) (2004) 227–253.

Mohamed El Menshawy is an adjunct professor at Concordia Institute for Information Systems Engineering, Concordia University, Canada, and an assistant professor in the Department of Computer Science, Menoufia University, Egypt. El Menshawy has a Ph.D. in Electrical and Computer Engineering from Concordia University obtained in 2012. His research interests are software engineering, social commitments, logic and formal methods, model checking, m-health services, mobile applications, and machine learning.

Jamal Bentahar received the bachelors degree in software engineering from the National Institute of Statistics and Applied Economics, Morocco, in 1998, the MSc degree in the same discipline from Mohamed V University, Morocco, in 2001, and the Ph.D. degree in computer science and software engineering from Laval University, Canada, in 2005. He is a Professor with Concordia Institute for Information Systems Engineering, Faculty of Engineering and Computer Science, Concordia University, Canada. From 2005 to 2006, he was a Postdoctoral Fellow with Laval University, and then NSERC Postdoctoral Fellow at Simon Fraser University, Canada. He is an NSERC Co-Chair for Discovery Grant for Computer Science (2016-2018). His research interests include the areas of computational logics, model checking, multi-agent systems, service and cloud computing, game theory, and software engineering.

Warda El Kholy is an associate researcher at Concordia Institute for Information Systems Engineering, Concordia University, Canada, and an assistant professor in the Department of Information System, Menoufia University, Egypt. El Kholy has a Ph.D. in Information and Systems Engineering from Concordia University obtained in 2016. Her research interests are social conditional commitments, logic and formal methods, model checking, communication protocols, web services, and web service composition.

Amine Laarej received his bachelor's degree in Software Engineering and Information Systems Modeling from the National Institute of Post & Telecommunication, Morocco, in 2015. He is currently pursuing his MSc degree in Quality Systems Engineering at Concordia University. His research interests include multi-agent systems, logics and formal methods, model checking, model based testing and software engineering.