Journal of Artificial Intelligence Research 57 (2016) 465-508 Submitted 03/16; published 11/16

ProMoca: Probabilistic Modeling and Analysis of Agents in Commitment Protocols

Akın Günay AKINGUNAY@NTU.EDU.SG
Yang Liu YANGLIU@NTU.EDU.SG
Jie Zhang ZHANGJ@NTU.EDU.SG
School of Computer Science and Engineering
Nanyang Technological University, Singapore

Abstract

Social commitment protocols regulate interactions of agents in multiagent systems. Several methods have been developed to analyze properties of commitment protocols. However, analysis of an agent's behavior in a commitment protocol, which should take into account the agent's goals and beliefs, has received less attention. In this paper we present ProMoca framework to address this issue. Firstly, we develop an expressive formal language to model agents with respect to their commitments. Our language provides dedicated elements to define commitment protocols, and model agents in terms of their goals, behaviors, and beliefs. Furthermore, our language provides probabilistic and non-deterministic elements to model uncertainty in agents' beliefs. Secondly, we identify two essential properties of an agent with respect to a commitment protocol, namely compliance and goal satisfaction. We formalize these properties using a probabilistic variant of linear temporal logic. Thirdly, we adapt a probabilistic model checking algorithm to automatically analyze compliance and goal satisfaction properties. Finally, we present empirical results about efficiency and scalability of ProMoca.

1. Introduction

Social commitments provide a formal framework to define, regulate, and reason about interactions of agents in multiagent systems (Singh, 1999). A commitment is made from a debtor to a creditor to bring about a condition. For instance, a merchant (debtor) is committed to a customer (creditor) to deliver the goods that are purchased by the customer. Every commitment has a state that changes according to some events. For instance, when the merchant delivers the purchased goods, her commitment to the customer becomes fulfilled. Commitments do not enforce agents to bring about certain events. Instead, they regulate agents by defining how events affect states of their commitments. In other words, agents decide on fulfilling or violating their commitments autonomously. For instance, the merchant may decide not to deliver the goods to the customer. However, this event result in violation of her commitment, which may have consequences (e.g., the merchant may be sanctioned and also loses reputation). Through regulation, commitments establish the desired level of interdependence among agents without interfering with their autonomy. Commitments can be combined to form commitment protocols, which can capture complex interactions among agents (Yolum & Singh, 2002).

Analysis of commitment protocols' properties is essential to ensure their effective operation. However, such analysis is challenging due to the rapidly increasing complexity of interaction in such protocols. Hence, development of efficient formal analysis methods for commitment protocols, which can cope with their complexity, is essential to create effective multiagent systems. Various formal properties of commitment protocols have been studied and several methods have been developed to analyze them (Yolum, 2007; Desai, Cheng, Chopra, & Singh, 2007a; Desai, Narendra, & Singh, 2008; El Menshawy, Bentahar, El Kholy, & Dssouli, 2013; El Kholy, Bentahar, Menshawy, Qu, & Dssouli, 2014).

However, analysis of an agent's properties when enacting a commitment protocol has received less attention (Marengo, Baldoni, Baroglio, Chopra, Patti, & Singh, 2011; Günay & Yolum, 2013; Kafalı, Günay, & Yolum, 2014). Such analysis aims to verify formal properties of an agent's behavior with respect to a commitment protocol (e.g., compliance of an agent's behavior with a protocol), which is crucial for developing effective agents. A key challenge of analyzing an agent's behavior with respect to a commitment protocol is uncertainty, which naturally occurs in multiagent systems due to several factors. One major factor is agent autonomy, which mainly corresponds to epistemic uncertainty. Specifically, agents act autonomously to pursue their own private goals. Hence, an agent cannot be certain about behaviors of other agents. Similarly, an agent's lack of awareness about its environment, which may be a result of limited sensory and reasoning capabilities, leads to epistemic uncertainty. Furthermore, many physical systems involve irreducible aleatory uncertainty that occurs due to physical variability present in an agent's environment. Agents cope with uncertainty by utilizing reasoning methods that can use potentially wrong or incomplete beliefs instead of exact knowledge (Halpern, 2003).

Most of the previous work on commitments do not address uncertainty. To fill this gap, in our previous work, we developed an analysis method for commitment protocols using probabilistic model checking, which can handle uncertainty in behaviors of agents (Günay, Songzheng, Liu, & Zhang, 2015). Our method uses an abstract formalism (specifically, a probabilistic automaton) for modeling and analyzing behaviors of agents in commitment protocols. However, from practical point of view, manual definition of commitment protocols and behaviors of agents in such an abstract formalism is a time consuming and error prone task. Besides, it requires a modeler to have knowledge and expertise on the specific abstract formalism. Because of these issues, use of an abstract formalism for modeling is not adequate for practical development settings (e.g., when a developer verifies her own agent's implementation). The adequate approach is to use an expressive high level formal language for modeling, which can be automatically translated into an abstract formalism for formal analysis. To the best of our knowledge, there is no such dedicated formal modeling language to capture uncertainty of agents in the context of commitment protocols.

In this paper we present ProMoca framework to address this issue. ProMoca provides an expressive modeling language that includes various language elements to model commitment protocols, and also various aspects of agents, such as different goal types, beliefs, and behaviors. Besides, ProMoca supports probabilistic modeling to capture uncertainty in behaviors and beliefs of agents. In ProMoca we also pay special attention to two essential properties of an agent's behavior with respect to a commitment protocol. The first of these properties is compliance of an agent's behavior with a commitment protocol. That is, whether an agent's behavior fulfills the agent's commitments. The second property considers whether an agent's behavior satisfies the agent's goals in the context of a commitment protocol. Since agents are interdependent, neither compliance nor goal satisfaction can be analyzed just considering an agent's own behavior in isolation. To cope with interdependence, ProMoca uses an agent's beliefs about other agents' behaviors. ProMoca adopts our previous probabilistic model checking method (Günay et al., 2015) for analyzing these properties. To evaluate efficiency and scalability of ProMoca we use three realistic examples. Besides, we compare ProMoca's performance with PRISM, which is a state-of-the-art probabilistic model checker. Our main contributions are as follows:

- We develop the new modeling language ProMoca (we use the name ProMoca both for the modeling language and the analysis framework interchangeably). We define formal syntax and operational semantics of ProMoca, which provides dedicated language elements to define and manage a commitment protocol. ProMoca also provides various language elements to define behaviors, goals, and beliefs of an agent in the context of the commitment protocol. Moreover, ProMoca provides probabilistic and non-deterministic elements to model uncertainty in an agent's beliefs. To the best of our knowledge, ProMoca is the first formal language that provides dedicated elements for modeling commitment protocols, agents, and uncertainty in a unified environment.

- We identify and develop formal definitions of compliance and goal satisfaction properties. Different than the previous definitions of these properties in the literature, in our formalization we take uncertainty of an agent's beliefs about other agents' behaviors into account using a probabilistic variant of linear temporal logic. Accordingly, we can define these properties in a more precise and flexible manner.

- To analyze an agent's behavior in the context of a commitment protocol by verifying compliance and goal satisfaction, we define the complete ProMoca framework, in which modeling is done by using our new formal language that we develop in this paper, and analysis is done by adopting our previously developed probabilistic model checking method (Günay et al., 2015).

- We substantially extend our earlier preliminary experimental result (Günay et al., 2015), by conducting a detailed empirical study to validate ProMoca's practical usefulness and scalability in three well-studied scenarios from the literature, namely aerospace aftercare, international insurance (Jakob, Pěchouček, Miles, & Luck, 2008), and NetBill (Sirbu & Tygar, 1995). Our results show that ProMoca outperforms the state-of-the-art general purpose probabilistic model checker PRISM when verifying compliance and goal satisfaction properties of an agent's behavior in the context of a commitment protocol.

This paper is organized as follows. In Section 2, we introduce the fundamental concepts about commitment protocols, agents, and analysis of their behaviors. In Section 3, we define ProMoca's formal syntax and operational semantics. In Section 4, we define our analysis framework and model checking algorithm. In Section 5, we present an empirical evaluation of our framework. Section 6 provides a survey of related work. Finally, in Section 7, we conclude the paper with a discussion of our approach and listing our future directions.

2. Background: Commitments, Agents, and Analysis

In this section we provide an overview of commitments, and agent concepts, such as behaviors, goals, and beliefs, which are key to ProMoca. We also discuss compliance and goal satisfaction properties with respect to these concepts to motivate our research.

2.1 Commitments

A commitment is made from one agent to another to bring about a condition (Singh, 1999). Conventionally, a commitment is denoted by **C**(*debtor*, *creditor*, *antecedent*, *consequent*) in which *debtor* and *creditor* are agents, and *antecedent* and *consequent* are conditions. Intuitively, a commitment means that the debtor is committed to the creditor to bring about the consequent, if the antecedent holds. For instance, the commitment **C**(*merchant*, *customer*, *goods-purchased*, *goods-delivered*) captures the merchant's commitment to the customer to deliver some goods (represented by the proposition *goods-delivered*), if the goods are purchased by the customer (represented by the proposition *goods-purchased*).

The exact meaning of a commitment depends on the type of the condition that is used as its consequent. Achievement conditions are the most widely used type for a commitment's consequent. For brevity, we call a commitment with such a condition simply as an achievement commitment. The above commitment of the merchant to deliver the purchased goods to the customer is an example of an achievement commitment. Use of a maintenance condition for a commitment's consequent has been also considered in the literature (Fornara & Colombetti, 2002; Mallya & Huhns, 2003; Günay & Yolum, 2011; Chesani, Mello, Montali, & Torroni, 2013). A commitment with a maintenance condition as its consequent is fulfilled, if the condition is maintained until another termination condition occurs. We call a commitment with such a condition simply as a maintenance commitment. For instance, an internet service provider may be committed to a customer to provide internet connection for a month, if the customer purchases a data plan from the internet service provider.

Commitments can also be considered in a subscription model. For instance, instead of purchasing a data plan for a single month, a customer may subscribe to a data plan for a year in a monthly basis (i.e., duration of each period in the subscription is a month). As a result, the internet service provider has a separate commitment for each month of the year to provide internet connection to the customer, as long as the customer pays the corresponding monthly subscription fee. In technical terms, the subscription itself can be considered as a template (e.g., if the customer pays the subscription fee for a specific month, the service provider becomes committed to provide internet connection for that month) for creating the concrete commitment instances for each period of the subscription. For example, we create twelve commitment instances from the above template (one for each month) by setting each commitment's antecedent and consequent to propositions that model payment of subscription fee and provision of internet for a specific month, respectively. To the best of our knowledge, such a subscription model is not formally considered in the previous research. However, subscriptions are part of many real world settings. Accordingly, we formalize them in ProMoca. Note that the subscription model can be considered as a special case of the meta-commitment concept (Chopra & Singh, 2015, 2016a), which is more general since it is not bound by the specification of a subscription.

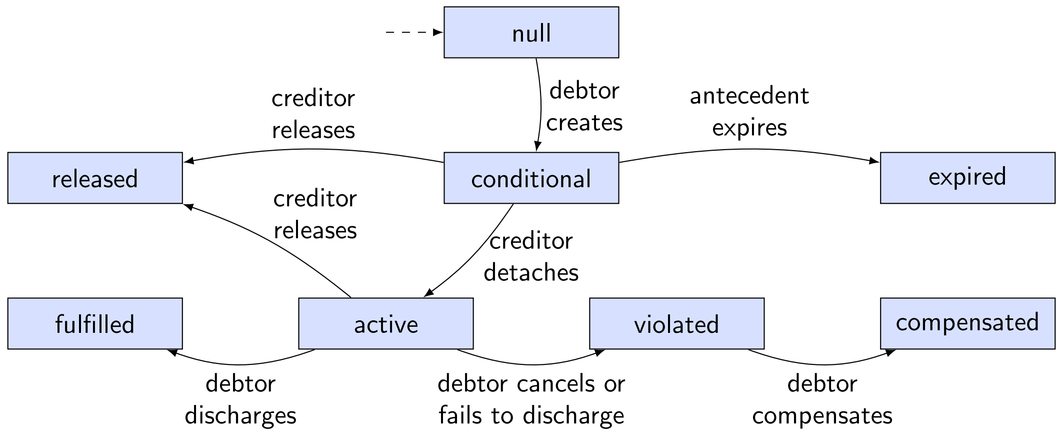

Figure 1: The lifecycle of a commitment.

A commitment's state evolves over time according to public events (i.e., they are independent from an agent's internal state). The lifecycle of a commitment has been studied extensively in the literature (Yolum & Singh, 2002; Fornara & Colombetti, 2002; Chesani et al., 2013). Here, we use the commitment lifecycle that we show in Figure 1, where the labels of the rectangles represent commitment states, and the edge labels show events with corresponding agents who trigger the state changes.

A commitment is in *null* state before its creation. A commitment is created by its debtor in *conditional* state. In this state neither the antecedent nor the consequent of the commitment holds. For instance, the commitment from the merchant to the customer is created by the merchant (i.e., debtor) and initially there is neither a payment nor a delivery. Hence, the commitment is conditional. The intuitive meaning of a conditional commitment is similar to an offer that states, if the antecedent starts to holds, the debtor becomes committed to the creditor to bring about the consequent.

If a commitment's antecedent is not satisfied, the commitment becomes *expired*. The creditor of a conditional commitment may also explicitly release the debtor from its commitment, which makes the commitment *released*. Expired and released states are terminal states. If the antecedent of a conditional commitment is satisfied, the creditor detaches the commitment, and the commitment becomes *active*. Detachment of a commitment is independent from how its antecedent is satisfied. That is, the antecedent may be satisfied either by an event that occurs as a direct result of an action of the creditor (e.g., customer pays), or by an external event (e.g., the customer's bank may pay on behalf of the customer). An active commitment intuitively means that the debtor is committed to bring about the consequent of the commitment.

Fulfillment and violation of a commitment depends on the type of the commitment's consequent. If the consequent of an active achievement commitment is satisfied, the debtor immediately discharges the commitment, and the commitment becomes *fulfilled* (e.g., the merchant delivers). A maintenance commitment is fulfilled, if the consequent of the commitment is maintained until another condition that determines the termination of the commitment occurs (e.g., the internet service provider maintains internet connection of the user until the termination of the data plan). As in the case of the detachment, a commitment's fulfillment is also independent from how the consequent is satisfied (e.g., the merchant herself may deliver, or she could delegate the delivery to a courier). A fulfilled commitment intuitively means that the debtor has honored her responsibility, and fulfilled state is terminal. As in the case of a conditional commitment, a creditor may also release a debtor from its active commitment.

If either the consequent of an active commitment is not satisfied, or the debtor explicitly cancels her commitment, the commitment becomes *violated*. For an achievement commitment, failure of the consequent may depend on another condition (e.g., a deadline). For a maintenance commitment, failure of the consequent occurs, if the consequent condition does not hold at any moment before the termination condition of the commitment holds. Note that due to autonomy of agents, a debtor may intentionally choose to violate a commitment, even if she can fulfill it. For instance, the merchant may decide to sell the goods to another customer for better profit, and may violate her commitment to the original customer. On the other hand, the debtor may also violate a commitment unintentionally. For instance, the merchant may fail to deliver because of bad weather conditions. In any case, a violated commitment intuitively means that the debtor failed to honor her responsibility. Depending on the domain of application, a violated commitment can be compensated by the debtor by taking a certain action, which makes the commitment state *compensated* (Torroni, Chesani, Mello, & Montali, 2010; Kafalı & Torroni, 2012; Chopra & Singh, 2015). For instance, if the merchant fails to deliver the goods to the customer, she violates her commitment. However, she can refund the customer to compensate her commitment's violation. Compensation allows agents to restore their interaction back to a desirable state, when it is interrupted due to a violated commitment. Compensated state is terminal. If a commitment is violated and there is no way for compensation, violated state is counted as terminal.

Note that a subscription does not have a state by itself, since it does not define a concrete commitment, but it only provides a template for instantiating concrete commitments for each period of the subscription. Hence, the notion of state for a subscription is captured in an abstract manner by the states of corresponding concrete commitment instances.

In many applications, agents engage complex interactions that cannot be represented by a single commitment. To capture all the aspects of such complex interactions, multiple commitments are considered together as a commitment protocol (Yolum & Singh, 2002). For instance, while the merchant's commitment to the customer captures the payment and delivery aspects of their interaction, another commitment $\mathbf{C}(\mathit{merchant}, \mathit{customer}, \mathit{goods\text{-}defective}, \mathit{goods\text{-}replaced})$, which states that the merchant is committed to the customer to replace a defective good, captures the warranty related aspects of their interaction. These two commitments (and other prospective commitments) form a commitment protocol.

2.2 Agents and Analysis of their Behaviors

In this paper, our main objective is to develop a dedicated formal language for modeling and analyzing an agent's behavior in a commitment protocol. When it is not clear from the context, we call this particular agent as the *target agent* to distinguish it from other agents.

In our analysis we use three kinds of information, which are available to the target agent. The first kind of information is the target agent's goals. We divide goals into two types as achievement and maintenance goals as it is customary in the agent literature (van Riemsdijk, Dastani, & Meyer, 2009; Chopra, Dalpiaz, Giorgini, & Mylopoulos, 2010). Achievement goals model situations, where an agent aims to achieve a certain satisfaction condition. For instance, a customer may aim to possess some goods. Maintenance goals model situations where an agent aims to maintain a certain satisfaction condition. For instance, an internet service provider may aim to maintain its internet service up and running. Furthermore, both goal types can be considered as one-time or persistent goals. One-time goals are pursued only once, and in general, they are subject to preconditions. For example, if the customer needs a certain type of good and she does not possess it, only than she has a goal to possess the good. Similarly, the internet service provider aims to provide and maintain internet connection for a customer, only if the customer has subscribed for the service. One-time goals might also have a termination condition (e.g., a deadline). If the termination condition of a goal holds before its satisfaction condition, the goal fails. Contrary to one-time goals, persistent goals do not have pre or termination conditions, and they persist for the whole lifespan of an agent. In the case of a persistent achievement goal, an agent aims to satisfy a condition that may fail from time to time, but always restored again after a while. For instance, a merchant may aim to dispose obsolete goods (e.g., old versions of a mobile phone) from its warehouse. Occasionally, some obsolete goods may be kept in the warehouse (e.g., when a new version of the mobile phone first appears, it takes a while to dispose the old phones), but they should be disposed eventually. On the other hand, a persistent maintenance goals represents a condition that an agent aims to keep satisfied continuously during its whole lifespan without any precondition. For instance, the merchant might aim to maintain a positive bank balance during its whole lifespan.

The second kind of information, which we use in our analysis, is a target agent's behavior. We assume that a target agent's behavior is a persistent (i.e., non-terminating) computation, in which the agent uses a set of if-then type rules to decide on its next action (e.g., if the merchant has a commitment to make a delivery and the commitment is active, then she delivers). If there is uncertainty about the results of the agent's actions, the decision rules may include non-deterministic and probabilistic components to model uncertainty. Suppose that the merchant may fail to deliver on time with probability 0.1 depending on weather conditions. In this case, the if-then rule that defines the merchant's behavior would be: if the merchant is committed to deliver and weather condition is bad, then the merchant delivers on time (and fulfills her commitment) with 0.9 probability, and fails to deliver on time (and violates her commitment) with 0.1 probability. As we demonstrate in the rest of this paper, combination of if-then rules, non-determinism, and probabilistic choice, in the context of a persistent computation provides us a rich and flexible model to define various complex agent behaviors.

The last kind of information that we use in our analysis is the target agent's beliefs about behaviors of other agents. Uncertainty is a natural element of this kind of information, since other agents are autonomous. Accordingly, we define the target agent's beliefs about other agents' behaviors in a similar manner to the target agent's own behavior using probabilistic choice and non-determinism. For instance, suppose that the merchant is dependent on a courier to make a delivery, which is necessary to fulfill her commitment to a customer. In such a situation the merchant may believe that the courier successfully delivers on time with probability 0.95 (and fails to deliver on time with probability 0.05). As we demonstrate later, we capture such a situation using a probabilistic choice in the beliefs of the target agent in a similar manner to our earlier example. Note that, in this paper we assume that probability values in an agent's behavior and beliefs are set by a modeler. These values may reflect intuition of the modeler, or they can be obtained from a statistical model (e.g., previous delivery results of the merchant or courier in different weather conditions).

Considering these three kinds of information, our objective is to analyze two key properties about a target agent's behavior in a commitment protocol. The first property is compliance, which holds if a target agent's behavior fulfills all of its active commitments in a protocol. A relaxed version of compliance may take compensation into account. That is, an agent complies with a commitment protocol by fulfilling its active commitments and also by compensating its violated commitments. The second property is goal satisfaction, which holds if a target agent satisfies its goals by enacting a commitment protocol. A target agent's beliefs about other agents play a crucial role in analyzing both properties since there is interdependence among agents. In other words, other agents' behaviors directly affect target agent. For instance, when the merchant relies on a courier for delivery, the merchant's compliance with the protocol and goal satisfaction cannot be correctly verified without taking the courier's behavior into account. In fact, if we consider only the merchant's own behavior, neither compliance nor goal satisfaction hold for the merchant, since she does not have delivery capability. However, if the merchant believes that the courier delivers with a high probability, then we can conclude that the merchant complies with the protocol and also satisfies her goal by enacting this protocol.

2.3 Running Example

In the rest of the paper we use a running example from aerospace aftercare domain, which is introduced by Jakob et al. (2008), and used in the literature for the evaluation of commitments and other normative models (Modgil, Faci, Meneguzzi, Oren, Miles, & Luck, 2009; Desai, Chopra, & Singh, 2009). In this example scenario, there is a manufacturer that provides aircraft engines to airline operators. When the manufacturer sells an engine to an airline operator, the manufacturer becomes responsible for keeping the engine operational by periodically servicing the engine. The airline operator should pay a service fee to the manufacturer. Besides, the airline operator should also provide operational data of the engine to the manufacturer, which is needed by the manufacturer to analyze the status of the engine. In order to service an engine, the manufacturer needs spare engine parts from several suppliers. However, the manufacturer can only use certain engine parts for servicing, which are approved by a monitoring aerospace agency. The airline operator may be monitored by different aerospace agencies depending on the regions in which the airline operates. Different agencies may have different policies about approved spare engine parts. Hence, use of a certain part for repair depends on the airline and the respective agencies. If the operational status of an engine is not maintained, the manufacturer should compensate this situation by paying a penalty to the airline operator. The manufacturer should also bring the engine back to the operational state within a certain amount of time. If the manufacturer fails to compensate, the contract may be canceled by the airline operator. We consider this example from the engine manufacturer's point of view to analyze its compliance and goal satisfaction.

3. Modeling of Commitment Protocols and Agents in ProMoca

We first present the formal syntax of ProMoca in Section 3.1. Then, we define the operational semantics of ProMoca in Section 3.2, and formalize compliance and goal satisfaction properties in Section 3.3. Finally, we discuss our key design decisions about ProMoca in Section 3.4.

3.1 Syntax

A ProMoca model is composed of three blocks, which define a set of global variables, a commitment protocol, and a target agent (i.e., the agent that we aim to verify). Below we define the syntax of each block with illustrative examples.

3.1.1 Global Variables

A ProMoca model includes a finite set of global variables, which are defined within the block **globals**{var; . . . var;}. Each variable var is defined using the keyword **variable** with the following syntax:

var ::= **variable** variable-name : {value-set} = value

where variable-name is the unique name of the variable, value-set is a finite set of strings that defines the domain of the variable (i.e., an enumeration of the variable's possible values), and value is the initial value of the variable, which must be an element of value-set. For instance, the status of an aircraft engine can be modeled using the following variable:

```
variable engine-status : {'unknown', 'operational', 'malfunction', 'maintenance'} = 'unknown';
```

The name of the variable is engine-status. The domain of the variable consists of four values, which capture different operational states of the engine. The initial value of this variable is 'unknown'.

3.1.2 Commitment Protocol

A commitment protocol is a composition of a finite set of commitments. The commitment protocol is defined within the block **protocol**{comm; . . . comm;}. Each commitment comm is defined using the keyword **commitment** and the following syntax:

comm ::= **commitment**(commitment-id, commitment-type, subscription-period,
debtor-id, creditor-id, antecedent, expiration, consequent, termination)
{observers}[commitment-id]

The first parameter commitment-id is the unique string identifier of the commitment. The type of the commitment is defined by commitment-type, which is either achievement or maintenance. Subscriptions are modeled using the subscription-period parameter, which is a finite integer greater than 0 that represents the total number of the subscription's periods. If the commitment is not a subscription, this parameter can be omitted, and ProMoca sets its value to 1 by default. The parameters debtor-id and creditor-id are the identifiers of the commitment's debtor and creditor, respectively. The parameters antecedent, expiration, consequent, and termination are expressions to define the antecedent, expiration, consequent, and termination condition of the commitment, respectively. The optional parameter observers is a set of agent identifiers that includes the agents, who can observe the state of the commitment. By default, the debtor and the creditor of a commitment can always observe the state of their commitment, and it is not required to list them as observers. If there are no observers other than the commitment's debtor and creditor, observers can be omitted. The last optional parameter commitment-id is the identifier of another commitment that can be used to compensate violation of the commitment. If there is no compensation of the commitment's violation, this parameter can be omitted.

An expression (e.g., antecedent parameter of a **commitment**) is a logical expression that is a compositions of conjunctions and disjunctions over global variable comparisons. Formal syntax of an expression is as follows:

```
expression   ::=  expression and expression | expression or expression | comparison
comparison   ::=  variable-name == value | variable-name != value | const
const        ::=  TRUE | FALSE
```

In an expression, we use the standard semantics of the logical operators **and** and **or**. Similarly, in comparison we use the standard "equal to" and "not equal to" semantics for operators == and !=, respectively. TRUE and FALSE are the standard boolean constants. Parentheses can be used regularly to define precedence of the logical operators, which we omit in the formal syntax for brevity.

ProMoca automatically creates several variables to capture the lifecycle of a protocol's commitments. Firstly, ProMoca creates a status variable for each commitment in the protocol. The names of these variables are automatically set by ProMoca using the <commitment-id>-state pattern (e.g., for a commitment with commitment-id 'c-1', the corresponding variable's name is c-1-state). Each such status variable has the domain {'null', 'conditional', 'active', 'fulfilled', 'violated', 'expired', 'released', 'compensated'} to capture the corresponding state of the commitment. These variables can be used in a ProMoca model as read-only variables. Secondly, ProMoca creates a subscription counter for each commitment to track the fulfillment of subscriptions. However, these counters are internal to ProMoca, and they cannot be directly accessed within a ProMoca model. The semantics of these variables are formalized later in Section 3.2.

Now we present some example commitments from the aerospace aftercare scenario to illustrate use of ProMoca's commitment syntax. Let us start with the simple achievement commitment: "if the operator pays the price of an engine to the manufacturer, the manufacturer is committed to deliver the engine". Suppose that there are payment and delivery deadlines (defined as variables). The commitment can be observed by the aerospace agency. If the commitment is violated, it cannot be compensated. This commitment is written in ProMoca as follows:

```
commitment('c-1', achievement, 1, 'manufacturer', 'operator',
             engine-paid == 'done', payment-deadline == 'past',
              engine-delivered == 'done', delivery-deadline == 'past')
             {'aerospace-agency'};
```

In our second example we consider maintenance of an engine according to the commitment: "if the manufacturer delivers an engine to the operator, the manufacturer is committed to the operator to maintain the engine operational for a year", which we identify as c-2. If the manufacturer fails to maintain the engine operational, and violates c-2, it should compensate this situation by repairing the engine and also paying a penalty, which is defined in the compensating commitment c-3. Note that c-2 explicitly declares c-3 as its compensating commitment. This declaration is essential for correctly managing the lifecycles of these commitments as we show later in Section 3.2. Note that we use the variable contract in the commitments to capture whether the contract between the manufacturer and operator is still valid. If the contract between these parties is terminated (e.g., delivery of an engine is canceled), their commitments expire.

```
commitment('c-2', maintenance, 1, 'manufacturer', 'operator',
    contract == 'valid' and engine-delivered == 'done', contract == 'terminated',
    engine-status == 'operational' or engine-status == 'maintenance', end-of-year == 'past')
    {'aerospace-agency'}['c-3'];
```

```
commitment('c-3', achievement, 1, 'manufacturer', 'operator',
    contract == 'valid' and c-2-status == 'violated',
    contract == 'terminated' or c-2-status == 'fulfilled' or c-2-status == 'released',
    engine-status == 'operational' and penalty-paid == 'done',
    compensation-deadline == 'past')
    {'aerospace-agency'};
```

Before we continue, let us emphasize the different meaning of the termination condition in achievement and maintenance commitments. In an achievement commitment, the debtor is committed to bring about the consequent at some point before the occurrence of the termination condition. For example, in the achievement commitment c-1, the termination condition is the delivery deadline, and c-1 is fulfilled, if the engine is delivered at any point before this deadline. Otherwise, c-1 becomes violated. On the other hand, in a maintenance commitment, the debtor is committed to maintain the consequent at every point from the commitment's detachment until the occurrence of the termination condition. For example, in the maintenance commitment c-2, the termination condition is completion of one year after delivery of an engine, and fulfillment occurs if the engine's operational status is preserved from the detachment of the commitment until the completion of the year. Otherwise, the commitment becomes violated immediately.

To complement c-2, we need another commitment c-4 that defines monthly servicing of an engine using a subscription model as follows. After the delivery of an engine, the manufacturer is committed to the operator to service the engine in a monthly basis for a year, as long as the operator pays the monthly service fee and provides the engine usage reports. For brevity, we omit the observers and compensation for this commitment.

```
commitment('c-4-', achievement, 12, 'manufacturer', 'operator',
    contract == 'valid' and engine-delivered == 'done' and service-fee-paid-* == 'done' and
        engine-report-provided-* == 'done',
    contract == 'terminated' or fee-deadline-* == 'past' or engine-report-provided-* == 'failed',
    engine-serviced-* == 'done',
    engine-serviced-* == 'failed' or engine-serviced-* == 'late');
```

Since the commitment is modeled as a subscription for a year in a monthly basis, there are twelve instances of this commitment. However, some conditions of the commitment apply only to individual instances. For example, there is a separate payment of the service fee for each month. ProMoca provides * notation for the variables that correspond to such instance conditions. For example, service-fee-paid-* means that there are twelve variables (e.g., service-fee-paid-1, service-fee-paid-2, etc.) to model each separate payment of the service fee. The first instance of the commitment becomes active, if service-fee-paid-1 and the other conditions of the antecedent hold. Note that only some conditions are instance specific. Other conditions, such as the delivery of the engine (i.e., engine-delivered), are used by all instances referring to the same variable.

3.1.3 Target Agent Specification

The target agent specification is defined in the block **agent**[agent-id]{agent-spec}, where agent-id is the unique identifier of the target agent. The target agent specification agent-spec is composed of four parts. The first part is the definition of the target agent's finite set of local variables, which are enclosed by the block **locals**{var; ...var;}. Local variables are intended to model the internal state of the target agent. Hence, they cannot be used in the context of global elements, and accordingly they cannot be accessed by other agents. For instance, the antecedent and the consequent of a commitment cannot include a local variable of an agent. Local variables are defined using the global variable syntax.

The second part of the target agent specification is the definition of the target agent's goals, which are defined in the block **goals**{goal; ...goal;}. The syntax of goal is as follows:

```
goal     ::=  pa-goal | pm-goal | a-goal | m-goal
pa-goal  ::=  pagoal(satisfaction)
pm-goal  ::=  pmgoal(satisfaction)
a-goal   ::=  agoal(precondition, satisfaction, termination)
m-goal   ::=  mgoal(precondition, satisfaction, termination)
```

Persistent achievement and maintenance goals are denoted by pa-goal and pm-goal, respectively. In both goal types, satisfaction parameters are expressions that model satisfaction condition of the goal, which can include only global and local variable comparisons (i.e., no commitment-state variables are allowed in these conditions). For example, the persistent maintenance goal of the manufacturer to keep a positive bank balance while interacting with operators and suppliers can be modeled by the following goal:

```
pmgoal(bank-balance == 'positive');
```

One-time achievement and maintenance goals are denoted by a-goal and m-goal, respectively. For both goal types, precondition, satisfaction, termination are expressions that model the pre, satisfaction, and termination condition of the goal. The pre and termination conditions of both goal types can include global, local, and commitment state variable comparisons. However, the satisfaction condition for both goal types can include only global and local variable comparisons.

For example, the manufacturer's one-time goal to deliver an engine when a payment is made (captured by the 'active' status of commitment c-1) can be modeled using the following achievement goal.

```
agoal(c-1-state == 'active', engine-delivered == 'done',
      engine-delivered == 'failed' or engine-delivered == 'late');
```

Note that if the delivery of the engine fails or it is delivered later than the deadline (i.e., delivered == 'failed' or delivered == 'late'), the one-time goal becomes terminated.

The last two parts of the target agent specification are the definition of the target agent's own behavior and the target agent's beliefs about the other agents' behaviors. The target agent's own behavior is defined in the block **behavior**{behavior}. The target agent's beliefs about the other agents' behaviors are defined in the block **beliefs**{[agent-id]{behavior}; ...[agent-id]{behavior};}. A behavior (either the target agent's own behavior or a believed behavior of another agent) is defined according to the following syntax:

```
behavior  ::=  cont | stop |
               action-label{assign, ..., assign} -> behavior |
               commit{commitment-id} -> behavior |
               release{commitment-id} -> behavior |
               cancel{commitment-id} -> behavior |
               behavior <> behavior |
               [expression] behavior |
               (probability) behavior ...(probability) behavior
```

Let us explain the intuitive meaning of each element in the behavior syntax. The primitive behaviors **cont** and **stop** restarts and terminates the current behavior, respectively. We use action-label{assign; ...; assign;} to capture agent actions and their effects on the variables. Each action has a label action-label that is used only to improve readability. That is, the label of an action does not have any meaning and the same label can be used with different sets of assignments at different places of the model. The effects of an action are captured by assigning new values to a finite set of variables. Actions in ProMoca are atomic. Hence, all the assignments that occur as the result of an action, are done in the given order without interruption. The syntax of an assignment assign is as fallows:

```
assign ::= var-name = value
```

Beside domain dependent actions (e.g., delivering an engine), ProMoca also provides three meta-actions **commit**, **release**, and **cancel** that capture the corresponding commitment operations to alter the state of a commitment (i.e., commitment state variable). The action **commit** is used by a debtor to create a commitment, **release** is used by a creditor to release the debtor from its commitment, and **cancel** is used by a debtor to cancel her commitment. Other than these three, ProMoca does not provide any meta-action to manipulate the state of a commitment directly. Instead, the state of a commitment is captured internally by ProMoca according to the values of the commitment's parameters (e.g., consequent) as we define in the semantics of ProMoca.

A non-deterministic choice over behaviors is captured by behavior <> behavior. That is, either the first or the second behavior can be performed by the agent, and the decision is non-deterministic. [expression] behavior captures a guarded behavior (i.e., if-then rule). The guard condition expression is a logical expression (as defined before). If the guard condition holds, then the corresponding behavior is performed. Otherwise, the behavior becomes blocked until the condition holds. Note that arbitrary number of guarded behaviors can be combined using a non-deterministic choice to model complex decision procedures. If this is the case, one of the behaviors, for which the corresponding guard condition holds, is selected for execution in a non-deterministic manner. Finally, (probability) behavior ... (probability) behavior denotes a probabilistic choice among the possible behaviors. Each probability is a real value and the sum of all probabilities is equal to 1 for a probabilistic choice. Probabilistic and non-deterministic choices are the main elements of ProMoca to incorporate uncertainty into the modeling and analysis processes.

An important distinction between the **behavior** and **beliefs** blocks is the use of variable types. Global variables can be used (both for reading and assignment) in both blocks. However, local variables of the target agent can only be used in the **behavior** block, since they are private to the target agent. Local variables are not accessible either for reading or for assignment in the **beliefs** block, since this block models other agents' behaviors, which should not use the target agent's local variables. Accessibility of commitment state variables (always as read-only) depend on the role of the corresponding agent in the commitments. For instance, if the target agent is a debtor, creditor, or observer of a commitment, the corresponding state variable can be queried from the **behavior** block. Similarly, if another agent is a debtor, creditor, or observer of a commitment, the part of the **beliefs** block that corresponds to this agent's behavior, can query the state of a commitment.

Let us present a (partial) behavior example for the manufacturer as below, which is interpreted as follows. If a commitment of the manufacturer to service an engine (i.e., an instance of c-4) becomes active, the manufacturer may behave in three different ways, according to the current situation. If the manufacturer already has the necessary parts to service the engine (modeled by the local variable part-availability), the manufacturer services the engine on time with 0.99 probability. There is a small probability (0.01) of failure to complete the service on time, which violates c-4. When the manufacturer does not have the necessary parts to service the engine, it can order them either from the first or second supplier. The manufacturer believes that (as we demonstrate later) the first supplier mostly delivers ordered parts on time. The manufacturer also believes that the second supplier delivers ordered parts late almost all the time. Furthermore, the manufacturer believes that the second supplier may even completely fail to deliver ordered parts. Accordingly, the manufacturer prefers to work with the first supplier. However, if the first supplier is not approved by the aerospace agency for the particular operator, who owns the engine, the manufacturer works with the second supplier. This situation is captured in the guard statements. Note that in the case of late delivery of ordered parts, timely service of the engine is affected. That is, if a late delivery occurs, the manufacturer services on time only with 0.75 probability. Hence, there is a bigger risk for violating c-4. Finally, after servicing and engine, availability of the spare parts for the next service period is determined as a non-deterministic choice. That is, the manufacturer does not know in advance how many spare parts it needs, and therefore any possible result (e.g., running out of spare parts) should be considered.

```
[c-4-state == 'active'] {
  [part-availability == 'available'] {
    (0.99) service-on-time{engine-serviced = 'done'} ->
      all-parts-consumed{part-availability == 'unavailable'} <> cont
    (0.01) service-late{engine-serviced = 'late'} ->
      all-parts-consumed{part-availability == 'unavailable'} <> cont
  } <>
  [part-availability == 'unavailable' and supplier-1 == 'approved'] {
    [part-delivery == 'on-time']
      (0.99) service-on-time{engine-serviced = 'done'} ->
        all-parts-consumed{part-availability == 'unavailable'} <> cont
      (0.01) service-late{engine-serviced = 'late'} ->
        all-parts-consumed{part-availability == 'unavailable'} <> cont
    } <>
    [part-delivery == 'late'] {
      (0.75) service-on-time{engine-serviced = 'done'} ->
        all-parts-consumed{part-availability == 'unavailable'} <> cont
      (0.25) service-late{engine-serviced = 'late'} ->
        all-parts-consumed{part-availability == 'unavailable'} <> cont
    }
  } <>
  [part-availability == 'unavailable' and supplier-1 != 'approved' and supplier-2 == 'approved'] {
    [part-delivery == 'late']
      (0.75) service-on-time{engine-serviced = 'done'} ->
        all-parts-consumed{part-availability == 'unavailable'} <> cont
      (0.25) service-late{engine-serviced = 'late'} ->
        all-parts-consumed{part-availability == 'unavailable'} <> cont
    } <>
    [part-delivery == 'failed']
      service-failed{engine-serviced = 'failed'} -> cont
    }
  }
}
```

Lastly, below we demonstrate some beliefs of the manufacturer about the suppliers. Suppose that the interaction of the manufacturer with the first and second supplier is regulated by the commitments c-5 and c-6, respectively. These commitments state that if the manufacturer orders a batch of spare parts from a supplier, the supplier is committed to deliver the parts to the manufacturer. For brevity we do not show these commitments. As we have mentioned earlier, the manufacturer believes that the first supplier is more reliable. Precisely, the first supplier delivers ordered parts on time with 0.8 probability, and late with 0.2 probability. The manufacturer also believes that the second supplier is unreliable. Precisely, the second supplier delivers order parts late with 0.95 probability. Besides, with 0.05 probability, the second supplier fails altogether to deliver ordered parts.

```
beliefs {
  ['supplier-1'] {
    [c-5-state == 'active'] {
      (0.8) delivers-on-time{part-delivery = 'on-time'} -> cont
      (0.2) delivers-late{part-delivery = 'late'} -> cont
    }
  };
  ['supplier-2'] {
    [c-6-state == 'active'] {
      (0.95) delivers-late{part-delivery = 'late'} -> cont
      (0.05) fails-to-deliver{part-delivery = 'failed'} -> cont
    }
  };
}
```

3.2 Semantics

Now we define PROMOCA's operational semantics. We first define a PROMOCA model and a *configuration* that captures the global state of a PROMOCA model.

Definition 1 (PROMOCA Model). *A* PROMOCA *model is a tuple* $\mathcal{O} = (Var, V_{init}, C, G, B_{agn}|||B_{bel_1}|||\dots|||B_{bel_n})$ *where Var is the set of variables (i.e., the composed set of global, local, commitment state, and internal variables),* V_{init} *is the initial valuation of the variables in Var, C is the commitment protocol (i.e., a set of commitments), and G is the target agent's goal set.* $B_{agn}|||B_{bel_1}|||\dots|||B_{bel_n}$ *is the parallel composition of the interleaved behaviors of the target agent (represented by* B_{agn}*) and the other n number of agents (i.e.,* B_{bel_i} *represents the* i^{th} *agent's believed behavior).*

Definition 2 (Configuration). *A configuration is a tuple* (V, B)*, where V is a valuation of the variables in Var, and B is a behavior.*

We define the operational semantics of each behavioral element using firing rules, which operate over configurations. Below, Σ denotes the set of all (visible) agent actions, and τ denotes an internal (invisible) action. We use α to denote any action in Σ. We write $V \models \mathsf{expression}$ to denote that the logical expression expression evaluates to true for valuation

V. We start from the operational semantics of **cont**, which causes the current behavior B to continue from its initial location without making any changes to the valuation V. This case is handled by the invisible τ action, as it is defined by the rule [cont].

$$\frac{\textbf{cont} \text{ is encountered in } B}{(V, \textbf{cont}) \xrightarrow{\tau} (V, B)} \qquad \text{[cont]}$$

Rule [prefix] captures how an action α of a behavior B modifies the values of the variables in a valuation V.

$$\frac{}{(V, \alpha\{\mathsf{assign},\ \ldots,\ \mathsf{assign}\} \text{ -> } B) \xrightarrow{\alpha} (\mathit{update}(V, \{\mathsf{assign},\ \ldots,\ \mathsf{assign}\}, C), B)} \qquad \text{[prefix]}$$

For readability, we use the auxiliary *update* function to handle assignment of new values to variables when modifying a configuration. We present this function in Algorithm 1. In Algorithm 1 (and also in the other related algorithms) we slightly abuse our notation for readability as follows. We assume that each assignment assign is represented as a variable and value pair (i.e., (variable, value)). We use the bracket notation V[variable] to access to variable in valuation V. The internal state (c-id-state) and subscription (c-id-subscription) variables are available to the algorithm for each commitment with identifier c-id. We also omit the observer lists of commitments, since they are irrelevant to assignments.

Input arguments of the *update* function are the current valuation V, the set of assignments *assignments* that occur as the result of the action α, and the commitment protocol C. The function first updates the global and local variables of V according to *assignments* (lines 1–2). Then, for each commitment c in the protocol, the function updates the state of c (i.e., c-id) with respect to the updated values of the variables if necessary as follows (lines 3–22). If the commitment is conditional and the expiration condition of the commitment holds in the updated valuation, then the commitment expires and its subscription counter is updated by the auxiliary function *update-subscription*, which we present in Algorithm 2, and explain in detail later (lines 5–7). Otherwise, if the antecedent of the conditional commitment holds in the updated valuation, the commitment becomes active (lines 8–9). If neither of these conditions hold, the state of the conditional commitment does not change, which we do not show in Algorithm 1 for brevity.

If the commitment is active, it is necessary to consider its type (i.e., achievement or maintenance) to determine its next state. If the termination condition of an active achievement commitment holds in the updated valuation, the commitment becomes violated (lines 12–14). Otherwise, if the consequent of the commitment holds in the updated valuation, the commitment becomes fulfilled (lines 15–16). If neither of these conditions hold, the state of the active achievement commitment does not change. If the consequent of an active maintenance commitment does not hold in the updated valuation, the commitment becomes violated (lines 18–20). Otherwise, if the termination condition of the commitment holds in the updated valuation, the commitment becomes fulfilled (lines 21–22). If neither of these conditions hold, the state of the maintenance commitment does not change. Finally, *update* returns the updated valuation (line 23).

Subscription and fulfillment status of a commitment are handled by the auxiliary functions *update-subscription* and *fulfillment* as we define in Algorithms 2, and 3, respectively.

Algorithm 1: Function *update*(V, *assignments*, C) returns the updated valuation V.

```
input  : valuation $V$, set of assignments $assignments$, commitment protocol $C$
output: updated valuation $V$
1  foreach (variable, value) in $assignments$ do
2      $V$[variable] $\leftarrow$ value
3  foreach commitment(c-id, c-type, subs, deb, cre, ant, exp, con, ter)[c'-id] $c$ in $C$ do
4      if $V$[c-id-state] = 'conditional' then
5          if $V \models$ exp then
6              $V$[c-id-state] $\leftarrow$ 'expired'
7              $V \leftarrow update\text{-}subscription(V, c)$
8          else if $V \models$ ant then
9              $V$[c-id-state] $\leftarrow$ 'active'
10     else if $V$[c-id-state] = 'active' then
11         if c-type = achievement then
12             if $V \models$ ter then
13                 $V$[c-id-state] $\leftarrow$ 'violated'
14                 $V \leftarrow update\text{-}subscription(V, c)$
15             else if $V \models$ con then
16                 $V \leftarrow fulfillment(V, c, C)$
17         else if c-type = maintenance then
18             if $V \not\models$ con then
19                 $V$[c-id-state] $\leftarrow$ 'violated'
20                 $V \leftarrow update\text{-}subscription(V, c)$
21             else if $V \models$ ter then
22                 $V \leftarrow fulfillment(V, c, C)$
23 return $V$
```

When a commitment reaches to a terminal state (e.g., fulfilled) the internal subscription counter of the commitment should be updated. If the commitment is not a subscription (i.e., it is not a template commitment and enacted only once), this update has no effect. Otherwise, the update occurs as it is defined in Algorithm 2, which takes the current valuation V and the commitment c with the identifier c-id as input arguments. First, the internal subscription counter (i.e., c-id-subscription) of the commitment is increased by one (line 1). Then, if the incremented subscription counter is less than or equal to the total subscription period subs (i.e., some periods of the subscription have not been considered yet), the commitment that corresponds to the next period of the subscription becomes conditional (lines 2-4). Finally, the updated valuation is returned. Remember that the subscription counter of any commitment is initially set by ProMoca to 1. Hence, the condition at line 2 should include equal to case. Also note that a non-subscription commitment has a

Algorithm 2: Function *update-subscription*(V, c) returns the updated valuation V.

```
input  : valuation $V$, commitment $c$
output: updated valuation $V$
1 $V[\text{c-id-subscription}] \leftarrow V[\text{c-id-subscription}] + 1$
2 if $V[\text{c-id-subscription}] \leq \text{subs}$ then
3     s-id-state ← concatenate (c-id, c-id-subscription)
4     $V[\text{s-id-state}] \leftarrow$ 'conditional'
5 return $V$
```

default period limit of 1. Hence, the condition at line 2 always fails for non-subscription commitments as a result of increasing the subscription counter by one at line 1.

Algorithm 3 defines the auxiliary *fulfillment* function, which takes the current valuation V, the fulfilled commitment c with the identifier c-id, and the commitment protocol C as input arguments. The function first sets the state of the commitment (i.e., c-id-state) to 'fulfilled' (line 1). Then, it updates the subscription counter of the commitment (line 2). Lastly, if the fulfilled commitment c is the compensation of another commitment c' in the context of C, which is determined by the auxiliary function *compensates*, and the commitment c' is currently violated, the state of c' is set to 'compensated' (lines 3–5). Note that, if c is a subscription, it compensates c' only after all the periods of c are fulfilled (i.e., condition $\text{subs} < V[\text{c-id-subscription}]$ holds).

Algorithm 3: Function *fulfillment*(V, c, C) updates the valuation V with respect to fulfillment of commitment c.

```
input  : valuation $V$, commitment $c$, commitment protocol $C$
output: updated valuation $V$
1 $V[\text{c-id-state}] \leftarrow$ 'fulfilled'
2 $V \leftarrow$ update-subscription$(V, c)$
3 $c' \leftarrow$ compensates$(c, C)$
4 if $c' \neq$ none $\wedge V[\text{c'-id-state}] =$ 'violated' $\wedge\ \text{subs} < V[\text{c-id-subscription}]$ then
5     $V[\text{c'-id-state}] \leftarrow$ 'compensated'
6 return $V$
```

Note that, all these algorithms can be encoded trivially as a set of firing rules, which are similar to the ones that we use to define operational semantics of the other ProMoca elements. However, we prefer our algorithmic representation for readability. In practice, these algorithms can be implemented efficiently using index structures from the variables of a valuation to commitments that include an index variable in their conditions. However, we prefer to explain them in the presented (less efficient but more intuitive) iterative form for clarity.

We continue with the semantics of the meta-operations to create, release and cancel a commitment, which are shown in rules [commit], [release] and [cancel], respectively. Here we use auxiliary $update_{commit}$, $update_{release}$ and $update_{cancel}$ functions to handle change in the valuation of the corresponding commitment's state variable (as we do for assignments). For instance, if [commit] fires for a commitment with the identifier commitment-id, $update_{commit}$ function assigns the value 'conditional' to the variable commitment-id-state, sets the corresponding subscription counter to 1, and returns the updated valuation V.

$$\frac{V \models \text{ commitment-id-state == 'null'}}{(V, \textbf{commit}\{\text{commitment-id}\} \text{ -> } B) \xrightarrow{\alpha} (update_{commit}(V, \text{commitment-id}), B)} \quad \text{[commit]}$$

$$\frac{V \models \text{ commitment-id-state == 'conditional' } \textbf{or} \text{ commitment-id-state == 'active'}}{(V, \textbf{release}\{\text{commitment-id}\} \text{ -> } B) \xrightarrow{\alpha} (update_{release}(V, \text{commitment-id}), B)} \quad \text{[release]}$$

$$\frac{V \models \text{ commitment-id-state == 'active'}}{(V, \textbf{cancel}\{\text{commitment-id}\} \text{ -> } B) \xrightarrow{\alpha} (update_{cancel}(V, \text{commitment-id}), B)} \quad \text{[cancel]}$$

These three rules do not define how a configuration changes when the condition of a rule does not hold. For instance, [cancel] rule does not define how the configuration changes, if the commitment is canceled when it is not active. To handle these situations we define the following three rules, which ignore a commitment operation and do not change the current configuration, if the condition of the rule does not hold. For instance, in [!cancel], if a commitment that is not active, is canceled, the cancel operation is ignored and the commitment's current state is preserved (i.e., the current configuration does not change).

$$\frac{V \models \text{ commitment-id-state != 'null'}}{(V, \textbf{commit}\{\text{commitment-id}\} \text{ -> } B) \xrightarrow{\alpha} (V, B)} \quad \text{[!commit]}$$

$$\frac{V \models \text{ commitment-id-state != 'conditional' } \textbf{and} \text{ commitment-id-state != 'active'}}{(V, \textbf{release}\{\text{commitment-id}\} \text{ -> } B) \xrightarrow{\alpha} (V, B)} \quad \text{[!release]}$$

$$\frac{V \models \text{ commitment-id-state != 'active'}}{(V, \textbf{cancel}\{\text{commitment-id}\} \text{ -> } B) \xrightarrow{\alpha} (V, B)} \quad \text{[!cancel]}$$

Rules [non-1] and [non-2] capture the semantics of non-deterministic choice between the behaviors B_i and B_j. If there is a non-deterministic choice between B_i and B_j, either B_i or B_j is selected in a non-deterministic manner for progressing according to the rules [non-1] or [non-2], respectively.

$$\frac{}{(V, B_i <> B_j) \xrightarrow{\tau} (V, B_i)} \quad \text{[non-1]}$$

$$\frac{}{(V, B_i <> B_j) \xrightarrow{\tau} (V, B_j)} \quad \text{[non-2]}$$

Rule [guard] captures the semantics of guarded behaviors. If the expression guard that is associated to the behavior B is evaluated to true according to the valuation V, then the action α occurs and the model progresses to the configuration (V', B'). Otherwise,

B becomes blocked until the model progresses to another configuration, in which guard evaluates to true. This may happen, if an interleaving behavior assigns new values to some variables, which causes the model to progress to a new configuration, in which guard is evaluated to true. However, if there is no such interleaving behavior, the current behavior is permanently blocked (i.e., deadlocked).

$$\frac{V \models \mathsf{guard} \text{ and } (V, B) \xrightarrow{\alpha} (V', B')}{(V, [\mathsf{guard}]B) \xrightarrow{\alpha} (V', B')} \qquad \text{[guard]}$$

Lastly, we define operational semantics of interleaving behaviors' parallel composition, which is denoted by $|||$. Remember that ProMoca models both the target agent's own behavior and the target agent's beliefs about the other agents' behaviors in separate interleaving behaviors. We compose these behaviors into a single behavior to verify properties of the target agent's behavior as we explain in Section 4. We achieve this using rules [int-1] and [int-2]. Intuitively, if there are two interleaving behaviors B_i and B_j (e.g., the target agent's own behavior and a believed behavior of another agent), either B_i or B_j is executed independently from the other behavior.

$$\frac{(V, B_i) \xrightarrow{\alpha} (V', B_i')}{(V, B_i \mathbin{|||} B_j) \xrightarrow{\alpha} (V', B_i' \mathbin{|||} B_j)} \qquad \text{[int-1]}$$

$$\frac{(V, B_j) \xrightarrow{\alpha} (V', B_j')}{(V, B_i \mathbin{|||} B_j) \xrightarrow{\alpha} (V', B_i \mathbin{|||} B_j')} \qquad \text{[int-2]}$$

To clarify parallel composition of two interleaving behaviors, we present a simple example in Figure 2, which shows possible executions of the two interleaving behaviors B_i = {ia1{vi = '1'} -> ia2{vi = '2'}} and B_j = {ja1{vj = '1'} -> ja2{vj = '2'}} with respect to the initial valuation V = ((vi : '0'), (vj : '0')) for the variables vi and vj. Note that we omit **stop** statements at the end of the behaviors for brevity. Each box in Figure 2 corresponds to a composed configuration such that the first line is the valuation V and the rest is the interleaving behaviors B_i and B_j. Edges between boxes show transitions between configurations according to actions taken by the behaviors, which are shown as edge labels. Note that once an action is taken, it is removed from the corresponding behavior to indicate its completion. For instance, there are two possible actions that can be taken in the initial configuration, namely ia1 in B_i and ja1 in B_j. If ia1 is taken, the value of vi is set to '1', and B_i becomes {ia2{vi = '2'}} in the new configuration that we reach as the result of ia1. In the final configuration, the valuation V is ((vi : '2'), (vj : '2')), and both behaviors B_i and B_j are empty, hence they stop (i.e., all possible actions are taken).

Our firing rules define how a behavior progresses from one configuration to another in a ProMoca model. Now we define execution of a ProMoca model with respect to these rules as a Markov Decision Process (MDP) (Bellman, 1957), which is expressive enough to capture both probabilistic and non-deterministic interleaving behaviors.

Definition 3 (Markov Decision Process). *A Markov decision process is a tuple* $\mathcal{M} = (S, Act, \boldsymbol{P}, \iota_{init}, AP, L)$ *where*

- *S is a finite set of states,*

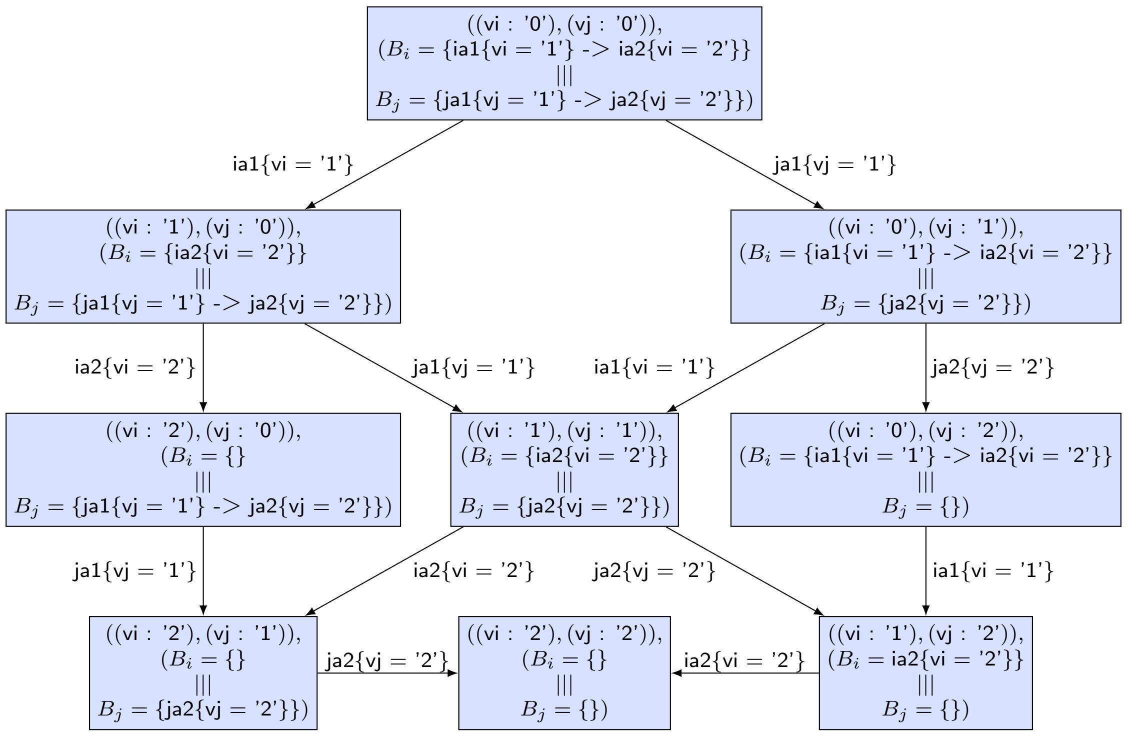
((vi : '0'), (vj : '0')),
(B_i = {ia1{vi = '1'} -> ia2{vi = '2'}}
|||
B_j = {ja1{vj = '1'} -> ja2{vj = '2'}})
ia1{vi = '1'}
ja1{vj = '1'}
((vi : '1'), (vj : '0')),
(B_i = {ia2{vi = '2'}}
|||
B_j = {ja1{vj = '1'} -> ja2{vj = '2'}})
((vi : '0'), (vj : '1')),
(B_i = {ia1{vi = '1'} -> ia2{vi = '2'}}
|||
B_j = {ja2{vj = '2'}})
ia2{vi = '2'}
ja1{vj = '1'}
ia1{vi = '1'}
ja2{vj = '2'}
((vi : '2'), (vj : '0')),
(B_i = {}
|||
B_j = {ja1{vj = '1'} -> ja2{vj = '2'}})
((vi : '1'), (vj : '1')),
(B_i = {ia2{vi = '2'}}
|||
B_j = {ja2{vj = '2'}})
((vi : '0'), (vj : '2')),
(B_i = {ia1{vi = '1'} -> ia2{vi = '2'}}
|||
B_j = {})
ja1{vj = '1'}
ia2{vi = '2'}
ja2{vj = '2'}
ia1{vi = '1'}
((vi : '2'), (vj : '1')),
(B_i = {}
|||
B_j = {ja2{vj = '2'}})
ja2{vj = '2'}
((vi : '2'), (vj : '2')),
(B_i = {}
|||
B_j = {})
ia2{vi = '2'}
((vi : '1'), (vj : '2')),
(B_i = ia2{vi = '2'}}
|||
B_j = {})

Figure 2: Possible executions of the interleaving behaviors B_i and B_j.

- *Act is a finite set of actions,*
- $\boldsymbol{P}$: $S \times Act \times S \mapsto [0, 1]$ *is a transition probability function such that for every state s in S and for every action α in Act:*

$$\sum_{s' \in S} \boldsymbol{P}(s, \alpha, s') \in \{0, 1\}$$

- $\iota_{init} : S \mapsto 1$ *is the initial distribution such that* $\sum_{s \in S} \iota_{init}(s) = 1$*,*
- *AP is a finite set of atomic propositions,*
- $L : S \mapsto 2^{AP}$ *is a labeling function.*

Without loss of generality we consider only finite MDPs (i.e., S, Act and AP are finite sets), and there are no terminal states in an MDP. Precisely, if $Act(s)$ denotes the set of enabled actions in s, then for any $s \in S$ it is the case that $Act(s) \neq \emptyset$. An action α is *enabled* in a state s if and only if $\sum_{s' \in S} \mathbf{P}(s, \alpha, s') = 1$. A state s' is an α-successor of another state s, if $\mathbf{P}(s, \alpha, s') > 0$. An MDP executes as follows. An initial state s_{init} is selected according to a stochastic experiment of the initial distribution ι_{init}. In any state s, first a non-deterministic choice is made between the enabled actions. Assume that action $\alpha \in Act(s)$ is selected. Then, an α-successor of s is selected randomly according to the distribution $\mathbf{P}(s, \alpha, \cdot)$.

Now we are ready to define the two necessary rules to systematically create an MDP from a ProMoca model using our operational semantics. Rule [non-prob] captures non-probabilistic cases. That is, for any non-probabilistic firing rule $(V, B) \xrightarrow{\alpha} (V', B')$, if α is performed from a configuration (V, B), the result is a single distribution μ that maps configuration (V', B') to 1.

$$\frac{(V, B) \xrightarrow{\alpha} (V', B') \text{ is a firing rule}}{(V, B) \xrightarrow{\alpha} \mu \text{ where } \mu((V', B')) = 1} \quad \text{[non-prob]}$$

Rule [prob] handles probabilistic choices, where the resulting distribution μ associates probability d_i with (V, B_i) for all i. Note that V remains unmodified during a τ transition.

$$\frac{}{(V, (p_1)B_1 \ldots (p_n)B_n) \xrightarrow{\tau} \mu \text{ where } \mu((V, B_i)) = p_i \text{ for all } i} \quad \text{[prob]}$$

Proposition 1. *For any* ProMoca *model* $\mathcal{O} = (Var, V_{init}, C, G, B_{agn}|||B_{bel_1}||| \ldots |||B_{bel_n})$ *there exists a corresponding MDP* $\mathcal{M} = (S, Act, \boldsymbol{P}, \iota_{init}, AP, L)$.

Proof: For any given ProMoca model $\mathcal{O}$, the following procedure constructs a corresponding MDP $\mathcal{M}$. Below, we use $Dom(var)$ to denote the domain of a variable $var \in Var$ and B to denote $B_{agn}|||B_{bel_1}||| \ldots |||B_{bel_n}$.

1. Creation of AP: For every value $val \in Dom(var)$ of every variable $var \in Var$ add a new proposition ap to AP. For instance, if the variable engine-status is an element of Var with the domain Dom(engine-status) = {'unknown', 'operational', 'malfunction', 'maintenance'}, add propositions *engine-status-unknown*, *engine-status-operational*, *engine-status-malfunction*, and *engine-status-maintenance* to AP, which correspond to the values 'unknown', 'operational', 'malfunction', 'maintenance' of engine-status, respectively.

2. Creation of S and L: Add a state s for every combination of the propositions in AP omitting combinations which include mutually exclusive propositions. A set of propositions are mutually exclusive (i.e., they cannot hold in the same state), if they are created from the domain of the same variable in Step 1. For instance, all the four propositions that are created from the domain of engine-status are mutually exclusive. Update the labeling L for each created state using the corresponding combination of propositions. For instance, considering the propositions in the first step, we create four states s_0, s_1, s_2, and s_3, and set the labeling function as follows: $L(s_0) = \{$*engine-status-unknown*$\}$, $L(s_1) = \{$*engine-status-operational*$\}$, $L(s_2) = \{$*engine-status-malfunction*$\}$, and $L(s_3) = \{$*engine-status-maintenance*$\}$. Note that there is a one-to-one correspondence between labels of states and valuations of a behavior's configurations. As a result, there is also a direct correspondence between states and configurations of a behavior.

3. Creation of ι_{init}: Set the ι_{init} such that the probability of the state, which has the labeling that corresponds to the valuation V_{init}, is 1, and probabilities of all other states are 0.

4. Creation of Act: For every assignment α in behavior B add a corresponding action to Act. Besides, for every commitment $c \in C$ add a set of meta-actions to Act, which correspond to the state changes of c.

5. Creation of $\mathbf{P}$: First, for each state $s \in S$ determine the set of enabled actions $Act^E_s \subseteq Act$. An action act is enabled in a state s (i.e., $act \in Act^E_s$) only if one of the following conditions hold:

    - act corresponds to an assignment α that occurs from the configuration $conf$, such that the valuation V of $conf$ corresponds to the labeling of the state s (i.e., $L(s)$). Note that the correspondence between a state and a configuration is already described in Step 2, and the correspondence between an action and an assignment is already described in Step 3. Intuitively, this condition captures the [prefix] rule. That is, if the next behavior element to apply in a configuration is an assignment, the action that corresponds to the assignment is enabled in the state that corresponds to the configuration.
    - act corresponds to a meta-action to change the state of a commitment, and act can be performed (according to the algorithms and rules which define the semantics of commitments) in the configuration $conf$, which corresponds to the to the labeling of the state s (i.e., $L(s)$). This condition captures the state changes of commitments.

    Then, for each state s, if Act^E_s includes one or more actions to change the state(s) of some commitment(s), a single transition with probability 1 is set to the destination state that labels the updated commitment states according to the algorithms and rules which define the semantics of commitments. The rest of the actions are ignored, if Act^E_s includes one or more actions to change the state(s) of some commitment(s). In other words, commitment states are updated instantaneously when needed, before any further agent action occurs. Otherwise, if there are no actions to update commitment states, a transition for each action $act \in Act^E_s$ with probability $p_{act}/\sum_{i=0}^{i=N} p_i$ is set for the destination states that are labeled according to the correspondences between the actions and the assignments, where p_i is the probability value of the action act_i as it is defined in the configuration $conf$ that corresponds to the state s, and N is the total number of actions in Act^E_s. □

Note that the procedure that we describe in the proof of Proposition 1 creates a finite MDP, since the number of variables (and their domains), and the number of assignments in a ProMoca model are finite. Also note that the actual computational complexity of this procedure occurs due to the last step, where we determine the transition probabilities of the MDP, which can be done in linear time with respect to the number of generated states in Step 2 and actions in Step 4. However, it is clear that the number of states of an MDP is exponential to the number of variables and their domains in the corresponding ProMoca model (see Step 1). This situation is commonly known as the state space explosion problem and constitutes the theoretical lower bound of the probabilistic model checking (Baier & Katoen, 2008).

3.3 Formal Properties

In this section we formalize the compliance and goal satisfaction properties using PLTL$_{\geq}$ (Baier & Katoen, 2008), which is a probabilistic variant of Linear Temporal Logic (LTL) (Pnueli, 1977). An LTL formula ϕ is defined over a set of atomic propositions AP using the temporal operators *globally* (G), *eventually* (F), and *until* (U). Satisfaction of an LTL formula is defined with respect to an infinite sequence of states $\varsigma = s_0, s_1, \dots$ as follows.

- $\mathsf{G}\,\phi$ holds in s_i, if ϕ holds in all future states s_j for $i \leq j$.
- $\mathsf{F}\,\phi$ holds in s_i, if ϕ holds at least in one future state s_j such that $i \leq j$.
- $\phi\,\mathsf{U}\,\phi'$ holds in s_i, if ϕ holds in all future states s_j until ϕ' holds in a future s_k, such that $i \leq j < k$.

PLTL$_{\geq}$ extends LTL with a probabilistic operator $\mathsf{P}_{\geq x}(\phi)$, where $x \in \mathbb{R}$ and ϕ is an LTL formula. Intuitively, $\mathsf{P}_{\geq x}(\phi)$ holds for an MDP model, if probability of satisfying ϕ is greater or equal to x. We define computation of this probability in detail in Section 4. In Sections 3.3.1 and 3.3.2, we use $\wedge$, $\vee$, $\Rightarrow$ and $\neg$ in PLTL$_{\geq}$ formulae for logical conjunction, disjunction, implication, and negation, respectively.

3.3.1 Compliance

Compliance of a target agent's behavior with a commitment protocol is the first property that we particularly aim to verify in ProMoca. Basically, if the target agent fulfills all of its active commitments in a commitment protocol, then the target agent's behavior complies with the commitment protocol. However, due to interdependence among agents and uncertainty about other agents' behaviors, it is usually not possible to determine an agent's compliance exactly. For instance, in the aerospace example, the manufacturer's compliance depends on the behaviors of the suppliers. Hence, we define compliance of a target agent's behavior with a commitment protocol in a relaxed manner with respect to a threshold θ_C that defines the minimal acceptable ratio of fulfillment for the target agent's active commitments. Precisely, for each commitment of the target agent, if a commitment becomes active, the commitment's fulfillment probability should be greater or equal to the threshold θ_C. Below, B is $B_{agn}|||B_{bel_1}|||\dots|||B_{bel_n}$.

Definition 4. *Given a* ProMoca *model* (Var, V_{init}, C, G, B) *and the corresponding MDP* $(S, Act, \boldsymbol{P}, \iota_{init}, AP, L)$*, the target agent's behavior* B_{agn} *complies with the commitment protocol* C *with respect to the threshold* θ_C*, if the following PLTL*$_{\geq}$ *formula holds:*

$$\forall\ \textbf{commitment}(\text{c-id, debtor, } \dots) \in C \text{ such that } \text{debtor} = agn :$$
$$\mathsf{P}_{\geq \theta_C}(\mathsf{G}(\text{c-id-state-active} \Rightarrow \mathsf{F}(\text{c-id-state-fulfilled} \vee \text{c-id-state-released})))$$

Intuitively, for every commitment of a protocol, in which the target agent is the debtor, if the commitment becomes active at any given state of the MDP, then the commitment's probability of finally becoming fulfilled or released in a future state should be greater or equal to the threshold θ_C. Note that in Definition 4, we omit commitment parameters (i.e., we instead use ...) that are irrelevant to compliance.

Even tough the threshold θ_C relaxes the requirements of compliance, Definition 4 is still a strict notion of compliance for some domains, since it requires fulfillment of every active commitment, and ignores compensations in the case of violation. However, as we have discussed in Section 2, compensation is a useful and widely used mechanism in many domains. Accordingly, we define *weak compliance*, which considers compensation, as follows:

Definition 5. *Given a* ProMoca *model* (Var, V_{init}, C, G, B) *and the corresponding MDP* $(S, Act, \boldsymbol{P}, \iota_{init}, AP, L)$, *the target agent's behavior* B_{agn} *weakly complies with the commitment protocol* C *with respect to the threshold* θ_C, *if the following PLTL*$_{\geq}$ *formula holds:*

$$\begin{gathered}\forall\ \textbf{commitment}(\text{c-id, debtor, } \dots) \in C \text{ such that } \text{debtor} = agn: \\ \mathsf{P}_{\geq \theta_C}(\mathsf{G}(\text{c-id-state-active} \Rightarrow \\ \mathsf{F}(\text{c-id-state-fulfilled} \vee \text{c-id-state-released} \vee \text{c-id-state-compensated})))\end{gathered}$$

Intuitively, for every commitment of the protocol, in which the target agent is the debtor, if the commitment becomes active at any given state of the MDP, then the commitment's probability of finally becoming fulfilled, released, or compensated in a future state should be greater or equal to the threshold θ_C.

Let us explain how compliance can be satisfied or failed on an example. Since computation of probabilities on an MDP is rather involved and not adequate for doing manually, we use a simplified case from our aerospace aftercare example, where the manufacturer can service an engine only if a supplier timely delivers the necessary parts for maintenance. Suppose that the manufacturer believes that the supplier delivers the parts on time with probability 0.9. Ignoring all other details, if the compliance threshold θ_C is set to 0.8, the verification process concludes that the manufacturer complies with the protocol, since the probability of timely servicing the engine is above of the threshold. On the other hand, if the compliance threshold θ_C is set to 0.95, the verification process concludes that the manufacturer fails to comply with the protocol, since the probability of timely servicing the engine is below of the threshold.

3.3.2 Goal Satisfaction

The second property that we aim to analyze in ProMoca is goal satisfaction, which defines whether a target agent can satisfy its goals by enacting a commitment protocol. However, as in the case of compliance, it is usually not possible to exactly determine goal satisfaction due to interdependence among agents and uncertainty in a multiagent system. Hence, as before, we define a threshold θ_S that defines the acceptable ratio of satisfaction for the goals of a target agent.

Definition 6. *Given a* ProMoca *model* (Var, V_{init}, C, G, B) *and the corresponding MDP* $(S, Act, \boldsymbol{P}, \iota_{init}, AP, L)$, *the target agent can satisfy its goals in* G *by enacting the commitment protocol* C *with respect to the threshold* θ_S, *if for each goal in* G, *the PLTL*$_{\geq}$ *formula that corresponds to the goal's type holds as below:*

- A persistent achievement goal can be satisfied, if the probability of achieving the satisfaction condition sat infinitely often is greater or equal to θ_S.

$$\forall\ \textbf{pagoal}(\mathsf{sat}) \in G : \mathsf{P}_{\geq \theta_S}\ \mathsf{GF}(\mathsf{sat}) \qquad \text{[pa-sat]}$$

- A persistent maintenance goal can be satisfied, if the probability of maintaining the satisfaction condition sat in every reachable state is greater or equal to θ_S.

$$\forall\ \textbf{pmgoal}(\mathsf{sat}) \in G : \mathsf{P}_{\geq \theta_S}\ \mathsf{G}(\mathsf{sat}) \qquad \text{[pm-sat]}$$

- A one-time achievement goal for which the precondition pre holds, can be satisfied if probability of not reaching the termination condition ter until the satisfaction condition sat is achieved, is greater or equal to θ_S. Intuitively, the goal should not be terminated until it is achieved.

$$\forall\ \textbf{agoal}(\mathsf{pre, sat, ter}) \in G : \mathsf{P}_{\geq \theta_S}\ \mathsf{G}(\mathsf{pre} \Rightarrow (\neg\ \mathsf{ter}\ \mathsf{U}\ \mathsf{sat})) \qquad \text{[a-sat]}$$

- A one-time maintenance goal for which the precondition pre holds, can be satisfied if probability of maintaining the satisfaction condition sat until reaching the termination condition ter is greater or equal to θ_S. Intuitively, the satisfaction condition should be maintained until the goal's termination.

$$\forall\ \textbf{mgoal}(\mathsf{pre, sat, ter}) \in G : \mathsf{P}_{\geq \theta_S}\ \mathsf{G}(\mathsf{pre} \Rightarrow (\mathsf{sat}\ \mathsf{U}\ \mathsf{ter})) \qquad \text{[m-sat]}$$

Let us explain how goal satisfaction can be satisfied or failed on a simplified case from our NetBill example, where the merchant has a persistent achievement goal to receive payments from customers. Suppose that the merchant believes that there is a customer who purchases a certain good every day with probability 0.5. Ignoring all other details, if the goal satisfaction threshold θ_S is set to 0.25, the verification process concludes that the merchant satisfies its goal, since the probability of receiving a payment on a daily basis is higher than the threshold. On the other hand, if the goal satisfaction threshold θ_C is set to 0.75, the verification process concludes that the merchant fails to satisfy its goal, since the probability of receiving a payment on a daily basis is lower than the threshold.

3.4 Remarks about PROMOCA Syntax and Semantics

While designing ProMoca's modeling language we are influenced from process algebra and communicating sequential processes (Hoare, 1978), which are successfully used for modeling various concurrent systems. However, in line with our objective of verifying properties of an agent's behavior in the context of a commitment protocol under uncertainty, we designed a novel language to model commitment protocols and agents, taking into account their goals, beliefs, and behaviors. One of our main motivations while designing ProMoca is to ease modeling of commitment protocols and agents by providing an intuitive language with elements to model them, which differs ProMoca from existing analysis tools. We discuss about ProMoca's modeling related benefits comparing to existing tools in more detail in Section 7. In the rest of this section we discuss some important aspects of ProMoca's modeling language.

A ProMoca model involves three types of variables that capture global, local and internal (commitment and subscription) state. This distinction among the variable types is used for type checking in the syntactic level to restrict the use of different variable types in certain language elements (e.g., commitments can include only global variables since they are public, and a target agent's beliefs about other agents' behaviors cannot include its local variables since they are private, etc.). On the other hand, in the semantic model of ProMoca, all variables are interpreted as global variables. It is straightforward to see that type checking of variables in the syntactic level is sufficient to correctly define and verify any PLTL$_{\geq}$ property, including compliance and goal satisfaction. Hence, it is safe to consider all variables as global variables in the semantic level as long as they are checked in the syntactic level.

As we discuss in ProMoca semantics, states of a commitment are determined by ProMoca according to evaluation of the commitment's conditions (e.g., consequent). Hence, ProMoca does not provide explicit meta-actions for most transitions (e.g., fulfillment). The exceptions are **commit**, **release**, and **cancel**. Initiation of a commitment is handled by **commit** as it is done by all previous work. Furthermore, explicit release and cancellation provides flexibility for modeling. For instance, many protocols involve some commitments to regulate exceptional situations. These commitments normally do not become active unless the corresponding exception occurs. Hence, they stay in conditional state even after the interaction of the involved parties has been completed. In such a situation, the creditors of such commitments can release the debtors from their conditional commitments. Cancellation can be used by a debtor to immediately terminate a commitment (by violating it) without waiting the occurrence of the commitment's termination condition. This is useful if the debtor realizes that it cannot fulfill its commitment. By canceling the commitment, the debtor gives time to the creditor to recover from the undesirable situation that occurs due to the violation of the commitment.

As a final remark on probabilistic modeling, note that probability computations in a ProMoca model may become rather complex. As our examples demonstrate, in many situations behaviors and belief may have arbitrary nesting and long sequences of actions, which involve non-deterministic and probabilistic choices. Furthermore, parallel composition of interleaving behaviors introduces additional complexity. As a result scalability of model checking may suffer when verifying compliance and goal satisfaction in large models. This is a common problem of probabilistic model checking. We address this issue in detail in the next two sections, and show that ProMoca can verify compliance and goal satisfaction in many realistic situations, although it is affected from scalability issues.

4. Verification

In this section we present the overall verification process of ProMoca, which is depicted in Figure 3. The inputs of the verification process are a ProMoca model, the type of the property (i.e., compliance or goal satisfaction) that is aimed to be verified, and a real value that corresponds to the threshold of the property (i.e., θ_C or θ_S). First, we create an MDP $\mathcal{M}$ from the input ProMoca model according to the operational semantics of ProMoca as in Section 3. In parallel, we extract the PLTL property ϕ (i.e., the property type and relevant propositions of the ProMoca model), which we will verify, according to

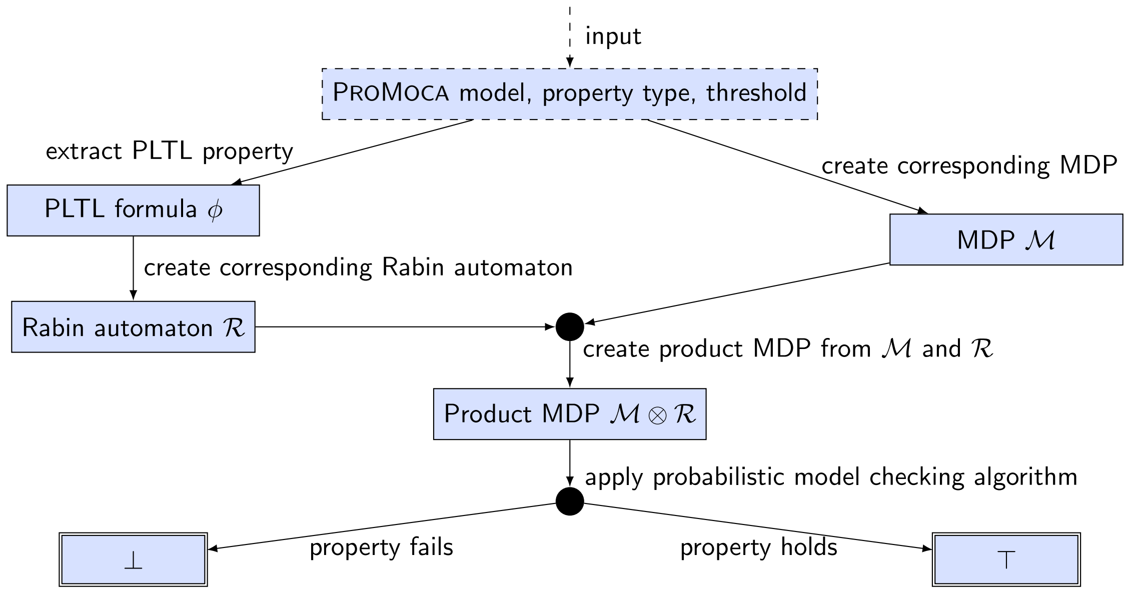
input
ProMoca model, property type, threshold
extract PLTL property
create corresponding MDP
PLTL formula ϕ
MDP $\mathcal{M}$
create corresponding Rabin automaton
Rabin automaton $\mathcal{R}$
create product MDP from $\mathcal{M}$ and $\mathcal{R}$
Product MDP $\mathcal{M} \otimes \mathcal{R}$
apply probabilistic model checking algorithm
$\perp$
property fails
property holds
$\top$

Figure 3: The overview of ProMoca's verification process.

the input. Then, we create a (deterministic) Rabin automaton $\mathcal{R}$ from ϕ using the standard translation of a PLTL formula into a Rabin automaton (Baier & Katoen, 2008). Once we have the MDP $\mathcal{M}$ and the Rabin automaton $\mathcal{R}$, we create their product $\mathcal{M} \otimes \mathcal{R}$, which is also an MDP (Baier & Katoen, 2008). In the last step, we use our probabilistic model checking algorithm that we present in Algorithm 4, on $\mathcal{M} \otimes \mathcal{R}$ to compute the satisfaction probability of ϕ in $\mathcal{M}$. If the result of this computation is greater or equal to the input threshold value, the verification process returns $\top$ to indicate that the property is satisfied. On the other hand, if the computed value is below the threshold, the verification process returns $\perp$ to indicate that the property is not satisfied.

When a property is not satisfied in non-probabilistic model checking, a trace (i.e., counterexample) is generated to explain why the failure occurs as a result of model checking. However, in probabilistic model checking there is no clear definition of such a counterexample since models are probabilistic, and properties are defined with respect to thresholds. Although, recent research in probabilistic model checking has addressed this issue and several approaches have been proposed (Andrés, D'Argenio, & Rossum, 2009; Han, Katoen, & Berteun, 2009; Schmalz, Varacca, & Völzer, 2009), ProMoca currently does not provide any functionality to generate a counterexample.

ProMoca uses the reachability checking algorithm that we have developed in our previous work (Günay et al., 2015). We present this algorithm in Algorithm 4 for completeness. The inputs of Algorithm 4 are an MDP $\mathcal{M} \otimes \mathcal{R} = (S, Act, \mathbf{P}, \iota_{init}, AP, L)$, that is the product of the MDP $\mathcal{M}$ and the Rabin automaton $\mathcal{R}$, which are extracted from the input ProMoca model, and $T \subseteq S$, the set of accept states of $\mathcal{R}$ as input. Algorithm 4 returns $P^R(T)$, which is the minimal probability of reaching the set of projected accept states T in the product MDP. In Algorithm 4, we use the auxiliary function $Pre(s)$ that returns the pre-states of a state s in an MDP. Formally, $Pre(s) = \{s' \mid \mathbf{P}(s, \alpha, s') > 0\}$. We use p_s to record the

Algorithm 4: Computation of reachability probabilities for target states.

```
input  : $\mathcal{M} \otimes \mathcal{R} = (S, Act, \mathbf{P}, \iota_{init}, AP, L),\ T \subseteq S$
output: $p_{s_0}$
1  let $S^{cur} \leftarrow T$ and $S^{pre} \leftarrow \emptyset$;
2  foreach $s \in S^{cur}$ do
3      let $p_s \leftarrow 1$;
4  while $S^{cur} \neq \emptyset$ do
5      foreach $s \in S^{cur}$ do
6          $S^{cur} \leftarrow S^{cur} \backslash \{s\}$;
7          foreach $s' \in Pre(s)$ do
8              $S^{pre} \leftarrow S^{pre} \cup \{s'\}$;
9              foreach $(s', t, \delta) \in Pr$ do
10                 let $p_n = 0.0$;
11                 foreach $s'' \in S$ such that $\delta(s'') > 0$ do
12                     $p_n \leftarrow p_n + \delta(s'') \times p_{s''}$;
13                 $p_{s'} \leftarrow Min(p_{s'}, p_n)$;
14     $S^{cur} \leftarrow S^{pre}$;
15     $S^{pre} \leftarrow \emptyset$;
16 return $p_{s_0}$;
```

probability of reaching T from s. Due to the existence of non-determinism in MDP, the result of reachability checking is a range instead of a single value. In Algorithm 4 we adopt a cautious approach and compute the *minimal* probability of reaching T.

The main idea of reachability checking is to start from target states and proceed backwards step by step while updating reachability probabilities of MDP states. Accordingly, first T' is assigned to S^{cur} (Line 1), which represents the current states in the iterative process, and the probabilities of these states are set to 1 (Lines 2-3). Then, a state s is removed from S^{cur} (Lines 5-6) and for each pre-state s' of s (Line 7) the probabilities are updated as follows: for each enabled transition t from s' with distribution δ, a variable p_n is created (Line 9-10) to record the probability of reaching T' from s' via δ. Afterwards, the sum of $\delta(s'') \times p_{s''}$ for all s'' satisfying $\delta(s'') > 0$ is assigned to p_n, i.e., p_n is the sum of the transition probabilities of this distribution times the corresponding successor state's probability to T' (Lines 11-12). To keep the minimal probability, $p_{s'}$ is set to the minimum value of $p_{s'}$ and p_n using Min function (Line 13). When all states in S^{cur} are considered, S^{cur} is set to pre-states of s (Line 14). The while loop at Line 4 terminates when no pre-states left (i.e., the minimal probability from s_0 to T is computed and stored in p_{s_0}). Finally, p_{s_0} is returned as the probability of reaching T from s_0 (Line 16).

5. Evaluation

We implement ProMoca framework in PAT (Process Analysis Toolkit) (Sun, Liu, Dong, & Pang, 2009), which is a state-of-the-art extendable model checking framework that provides various probabilistic model checking techniques such as PLTL model checking and reachability checking. To evaluate efficiency and scalability of ProMoca, we conducted a set of computational experiments. In these experiments we compared ProMoca's performance with PRISM (Kwiatkowska, Norman, & Parker, 2011), which is the most well-known probabilistic model checker. We used the following four examples in our experiments:

- AeroBase: In this example we use our aerospace aftercare case. However, we assume that the manufacturer always has sufficient supply of spare parts. Hence, we do not consider the interactions between the manufacturer and suppliers. The example involves a commitment to model the purchase of an engine and a commitment to model the overall maintenance agreement. Both of these commitments have also compensating commitments. Hence, there are four commitments to model the base terms of the agreement. Besides, for the whole duration of the agreement, there are two commitments for each month to model the servicing responsibility of the manufacturer (e.g., for a twelve month agreement there are 24 commitments). We consider this scenario from the manufacturer's perspective. The beliefs of the manufacturer involve probability of engine failure and probability of repairing a failed engine on-time.

- AeroFull: In this example, we use our aerospace aftercare case, but we do not assume that the manufacturer always has sufficient supply of spare parts. Hence, we consider a new commitment for each month of the agreement to model provision of supplies by the suppliers (e.g., there are 36 commitments for a twelve month agreement). Since, the manufacturer interacts with the suppliers, the beliefs of the manufacturer involve probability of successfully receiving spare parts from the suppliers.

- NetBill: In this example we model the well-known NetBill protocol (Sirbu & Tygar, 1995). NetBill provides a secure protocol for transactions between two parties (e.g., a merchant and a customer) in online environments. It is a standard protocol that is widely used in the commitment literature for evaluation. It involves three commitments to model offer, payment, and delivery phases of a transaction. In our experiments, we consider arbitrary number of transactions between a merchant and a set of customers from the merchant's perspective. The merchant's beliefs involve probability of the customers' acceptance for the merchant's offers.

- AGFIL: This example models a real world car insurance claim processing case (Desai et al., 2009). AGF Irish Life Holdings (AGFIL), a subsidiary of Allianz, is an insurance company in Ireland that underwrites car insurance policies. AGFIL cooperates with a call center and a consulting firm to process its policy holders' claims. A policy holder contacts the call center to make a claim. The call center forwards the claim to AGFIL. Then, AGFIL requests an investigation for the claim from the consultant to decide on the claim's validity. The consultant provides a report to inform AGFIL about the results of its investigation. Using the investigation report, AGFIL decides on whether to pay the claimed repair cost. This example includes three commitments to model the interactions of AGFIL with the other three stakeholders, namely, the policy holder, the call center, and the consultant. In our experiments, we consider arbitrary number of claims from a set of policy holders and verify properties of AGFIL's behavior. Beliefs of AGFIL involve probability of claim validity.

We run our experiments on a PC equipped with an Intel i7 3.0 GHz processor and 8GB RAM, running an 64-bit Windows 8 operating system. We conducted our experiments to verify both the compliance and goal satisfaction properties. However, in our experiments, we observed similar results for both properties. Hence, for brevity, we report our results only for the compliance property's verification. In Table 1 we report execution times (in seconds) of ProMoca and PRISM for verifying the compliance properties of the described examples. If the model checking process takes more than 1200 seconds, we report timeout (T/O). For each case, we reported average execution time of thirty runs to eliminate any spike in execution times that may occur due to use of resources by other processes in our system. We do not report variance of execution times since we observed negligible values.

To evaluate ProMoca's scalability with respect to different complexities of our examples, we use a control parameter in each example that determines the size of the corresponding ProMoca model. In the AeroBase and AeroFull examples this parameter is the number of months the engine manufacturer is committed to service an engine, which is represented by *mon* in Table 1. In the NetBill example the control parameter is the number of customers the merchant interacts, which is represented by *cus* in Table 1. In the AGFIL example the control parameter is the number of claims made by policy holders, which is represented by *cla* in Table 1.

Before discussing about our results in Table 1, let us explain the characteristics of the example models and how the described parameters affect them. In the aerospace examples, the outcome of a commitment that models a service agreement for a particular month, depends on the outcomes of the previous months' commitments. That is, if the manufacturer fails to service the engine in a previous month, there is a higher risk of engine failure for the current month, which may cause the manufacturer to violate its commitment to keep the engine operational. This is modeled in the manufacturer's beliefs. Accordingly, because of such dependencies, the models of the aerospace examples become substantially more complex when the duration of a service agreement (i.e., number of months *mon*) increases. Furthermore, the AeroFull example considers also the behaviors of the suppliers as part of the manufacturer's beliefs, which increases the complexity of the corresponding models even further. Hence, these examples capture complex realistic situations.

The models of NetBill and AGFIL examples are relatively simpler than the aerospace examples. We use this situation to evaluate ProMoca for cases where more than one commitment protocols are composed to have a more complex protocol. There are two possible compositions, namely sequential and parallel. We used NetBill example to examine parallel composition of commitment protocols. To this end, we increase the number of customers a merchant interacts in parallel from 1 to 10. That is, if there are 10 customers, the merchant enacts 10 instances of NetBill protocol in parallel. Parallel composition causes a model's size to grow exponentially. Hence, it has a significant impact on model complexity. We used AGFIL example to examine sequential composition of independent protocols. To this end, we increase the number of customer claims from 10 to 100. Each claim is processed sequentially one by one (i.e., first come first served) by AGFIL. Impact of independent sequential protocols on model complexity is substantially less than parallel composition of protocols, since each protocol instance can be verified independently. Therefore, model size growth is linear to the number of claims. Note that in Table 1, the first reported execution time for AeroBase example is when $mon = 6$. When *mon* is less than 6 both ProMoca

AeroBase			AeroFull			NetBill			AGFIL		
mon	ProMoca	PRISM	*mon*	ProMoca	PRISM	*cus*	ProMoca	PRISM	*cla*	ProMoca	PRISM
6	0.42	1.32	3	0.03	0.10	4	0.01 ≥	0.07	40	0.024	2.822
7	1.33	7.35	4	0.16	0.27	5	0.01	0.12	50	0.030	4.197
8	3.76	33.68	5	0.75	3.80	6	0.02	0.18	60	0.041	5.431
9	9.81	175.81	6	3.62	27.68	7	0.04	0.38	70	0.051	6.843
10	27.86	T/O	7	17.01	591.36	8	0.09	1.25	80	0.063	8.396
11	84.49	T/O	8	85.56	T/O	9	0.15	4.68	90	0.072	9.841
12	242.37	T/O	9	365.82	T/O	10	0.31	32.23	100	0.087	11.384

Table 1: Execution times of ProMoca and PRISM to verify compliance of the target agent in AeroBase, AeroFull, NetBill, and AGFIL examples.

and PRISM can verify compliance almost instantaneously in a few milliseconds. Similarly, we report our results in AeroFull, NetBill, and AGFIL examples for the control parameters starting from $mon = 3$, $cus = 4$ and $cla = 40$, respectively.

Table 1 indicates two main results. First, ProMoca clearly outperforms PRISM in terms of execution time for verifying an agent's compliance with its commitments under uncertainty. The main reason of this is the consideration of commitments as first-class objects in the syntax and semantics of ProMoca. Accordingly, ProMoca takes advantage of this dedicated representation, and uses several model checking techniques, which we have developed particularly for commitment protocols, for reducing model size to improve efficiency of model checking. For instance, ProMoca uses a specific partial order reduction technique that exploits dependencies among commitments to reduce the size of an MDP model's state space (Günay et al., 2015). Effects of this reduction technique is easy to see especially in the NetBill example, where there are a large number of parallel independent commitments in each NetBill protocol instance with a different customer. ProMoca effectively uses our dedicated partial order reduction technique to detect independence of commitments, and reduces model size, which provides a clear advantage in the NetBill example. In conclusion, our results show that development of dedicated model checking tools such as ProMoca is necessary for efficiently verifying an agent's behavior with respect to commitment protocols taking uncertainty into account.

Our second main result is usability of ProMoca in practical cases. In NetBill example ProMoca verifies compliance in milliseconds when ten protocols are executed in parallel. Similarly, in AGFIL example, ProMoca verifies the case, where 100 claims are considered, almost immediately. In AeroBase example, ProMoca verifies compliance even for the whole 12 month agreement within reasonable time. In the complete aerospace scenario, ProMoca verifies nine month agreement successfully. Note that given sufficient time and resources ProMoca can also verify a whole 12 month agreement. However, when a model involves large number of dependent commitments, and complex behaviors and beliefs, as our results show, ProMoca suffers from exponentially growing execution times, which is the result of rapidly growing model size in such situations. This is a well known issue of probabilistic model checking, which is unavoidable (Clarke, Grumberg, & Peled, 1999; Baier

& Katoen, 2008). Nevertheless, our comparative results with PRISM show that ProMoca is significantly more efficient than general purpose probabilistic model checkers for verifying the compliance and goal satisfaction properties of an agent in a commitment protocol.

6. Related Work

In this section we provide a non-exhaustive survey of related research. We start with previous work, which study various general properties of commitment protocols (i.e., independent from behaviors of enacting agents), and their verification. Yolum (2007) developed a framework to verify effectiveness, consistency, recovery, and fault-tolerance properties of commitment protocols. A commitment protocol is effective, if it is deadlock-free. It is consistent, if it does not involve conflicting propositions in commitment conditions. Finally, if at least one role of a commitment protocol is capable of taking a certain recovery action in the case of failure, the commitment protocol is recoverable. Besides, if any role is capable of recovering a commitment protocol, it is fault-tolerant. In addition to defining these properties, Yolum also provides a set of algorithms for their verification, which are based on analyzing states in arbitrary runs of commitment protocols. Desai et al. (2007a) also study deadlock-free and live commitment protocols. They develop several models of commitment protocols in PROMELA language of SPIN model checker (Holzmann, 2004), and define deadlock-freeness and liveness properties in LTL. Later, Telang and Singh (2012) use NuSMV model checker and Computation Tree Logic (CTL) to verify correctness of business patterns that are modeled as commitment protocols. Gerard and Singh (2013) introduce an approach to specify commitment protocols and their refinements through guarded messages. They implement their approach using MCMAS model checker.

Montali, Calvanese, and De Giacomo (2014) develop a data-aware framework using a first-order formalism to study impact of data that is available to agents, on commitments' evolution. They also show that a rich set of temporal properties in μ-calculus can be verified in their framework. El-Menshawy et al. (2013) develop ACTL^{*C}, which is an extension to CTL, by introducing a set of operators to model semantics of active commitments. The proposed semantics of the commitment operators are influenced by a previous proposal of Singh (2008). The paper shows that the proposed logic can be reduced to another logic GCTL*, which can be verified by CWB-NC model checker (Bhat, Cleaveland, & Groce, 2001). Later, El Kholy et al. (2014) propose another extension to CTL, which is called CTL^{CC}, to capture lifecycle of conditional commitments. They also extend the standard symbolic model checking algorithm of CTL in line with their proposal. Sultan, Bentahar, Wan, and Al-Saqqar (2014) propose PCTLC, which extends probabilistic CTL, to verify commitment protocols. PCTLC includes social operators to represent active commitments and their fulfillment. The proposed model checking technique consists of a set of reduction rules to reduce the PCTLC model checking problem to PCTL model checking, which are implemented on PRISM model checker.

Our proposal differs from these studies at several points. Firstly, none of these studies consider development of a modeling language as we do in ProMoca. They either use modeling languages of general purpose model checkers or an abstract formalism for modeling. Secondly, these studies do not consider behaviors of agents, since they aim to verify general properties of commitment protocols. Finally, probabilistic models are not considered in the previous work. The exception is the work by Sultan et al. (2014), which uses a probabilistic variant of CTL. However, Sultan et al. only consider active commitments. Besides, due to the use of CTL, they consider a different model checking algorithm than we use in ProMoca.

Some recent work aim to analyze a commitment protocol from an agent's point of view. Objectives of these studies are closer to the objective of ProMoca. Marengo et al. (2011) discuss agents' control on events in commitment protocols. Basically, an agent has control over an event, if the agent itself can initiate the event, or another agent that is capable to initiate the event is committed to do so. They also discuss a notion of safety. A commitment is safe for its debtor, if the debtor controls the antecedent and can avoid situations in which the commitment becomes active, or if the debtor controls the consequent and therefore able to fulfill the commitment when it becomes active. Marengo et al. develop REGULA framework to formalize control and safety properties, which also provides reasoning rules to evaluate systems for these properties. However, they do not develop a practical reasoner and also do not address uncertainty.

Günay and Yolum (2013) study feasibility of a commitment protocol for an agent, considering time constraints and resources requirements that should be satisfied by the agent to fulfill its commitments. A commitment protocol is feasible for an agent, if the agent owns sufficient resources, or the agent is the creditor of a set of commitments that provide the agent the resources to fulfill its own commitments on time. They, model verification of feasibility as a constraint satisfaction problem. Günay and Yolum discuss potential behaviors of agents and describe how unexpected behaviors (e.g., violations of commitments) may be handled. However, their consideration of behaviors do not address probabilistic models and requires manual configuration of their framework. Kafalı, Günay, and Yolum (2014) develop GOSU framework that provides a reasoning mechanism to determine goal support in a commitment protocol, which is similar to our goal satisfaction property. They model goal support as a reachability property and use reactive event calculus for reasoning (Chesani et al., 2013). Their approach considers only achievement goals, and avoid uncertainty by assuming that agents always honor their commitments.

Torroni and colleagues develop a monitoring framework for commitments using $\mathcal{S}$CIFF abductive logic programming proof-procedure (Alberti, Chesani, Gavanelli, Lamma, Mello, & Torroni, 2008; Chesani et al., 2013). Their main motivation is to develop a formal and operational framework that efficiently monitors commitments and verifies compliance of agents' actions with protocols that they enact. Since the main interest of this framework is run-time monitoring, Torroni and colleagues pay special attention to a commitment's time constraints. Although such constraints have been studied also by other researchers, Torroni and colleagues provide a concrete operational framework that can handle these constraints at run-time. For this purpose they utilize an event driven implementation of event calculus called reactive event calculus, which is based on the maximum validity interval concept of cached event calculus introduced by Chittaro and Montanari (1996). Use of reactive event calculus eliminates the necessity of backward reasoning at each event occurrence and accordingly it is possible to do reasoning efficiently at run-time. Their monitoring framework and ProMoca are complementary to each other. While ProMoca handles design-time issues, their framework addresses run-time monitoring.

Compliance is addressed also in the context of norms (e.g., prohibitions). Vasconcelos (2005) develops a declarative approach to analyze electronic institutions to determine whether agents commit and fulfill norms of institutions. Aldewereld, Vázquez-Salceda, Dignum, and Meyer (2006) develop a framework to verify norm compliance of agent behavior templates, which they call as protocols. They consider only sequential protocols and define norm compliance using LTL. A semi-automated theorem-proving approach is used for verification instead of model checking. Craven and Sergot (2008) develop $n\mathcal{C}+$ as an extension to the action language $\mathcal{C}+$ (Giunchiglia, Lee, Lifschitz, McCain, & Turner, 2004). $n\mathcal{C}+$ introduces two new forms of rules, namely state and action permission laws, for representing normative aspects of multiagent systems. Semantics of their language is defined with respect to colored labeled transition systems, which represent desired and undesired states, and also transition of a modeled multiagent system. By associating a subset of transitions with a particular agent's actions, they can verify compliance and other properties of agent behaviors. $n\mathcal{C}+$ provides a rich language to model multiagent systems. However, Craven and Sergot do not consider uncertainty and accordingly they do not provide probabilistic modeling and reasoning. Besides, norms are not defined with respect to a lifecycle as we do for commitments. Instead, norms are considered as (if-then) rules. Furthermore, they do not represent relations between different norms explicitly as we do, for instance for compensation. An interesting future direction is to investigate how these two formalism can be combined to develop a more expressive modeling and analysis environment.

In terms of model checking, the most relevant work to ProMoca is MCMAS, which is a state-of-the-art model checker dedicated to verification of multiagent systems (Lomuscio, Qu, & Raimondi, 2009). MCMAS uses Interpreted Systems Programming Language (ISPL) for modeling, which is based on the interpreted systems semantics (Fagin, Halpern, Moses, & Vardi, 2003). In ISPL, a multiagent system is modeled as a composition of a set of agents and their environment. Each agent is defined as a set of internal states using a set of private variables, and a protocol, which models the decision making mechanism of the agent. Agents interact through publicly observable actions. Local states of agents evolve according to an evolution function, which uses joint actions of agents. MCMAS supports verification of agent-oriented logics, such as Alternating-time Temporal Logic (Alur, Henzinger, & Kupferman, 2002) and epistemic operators (Fagin et al., 2003), using Ordered Binary Decision Diagrams (Bryant, 1986) and symbolic model checking techniques. There are several differences between MCMAS and ProMoca. While MCMAS is a general model checker for all types of multiagent systems, ProMoca focuses on verifying agent behaviors in commitment protocols. Accordingly, ProMoca provides dedicated language elements for defining commitment protocols. Moreover, ProMoca aims to verify an agent's behavior taking uncertainty of the agent's beliefs into account. To achieve this, ProMoca provides probabilistic modeling and reasoning capabilities for agent beliefs. ProMoca does not use interpreted systems semantics, since we do not aim to verify epistemic logic specifications. Finally, ProMoca uses an automata based approach instead of symbolic model checking, which is more appropriate for verifying our properties.

In the recent years, commitments have been used to model various practical situations. Desai, Chopra, Arrott, Specht, and Singh (2007b) provide a commitment-based solution for formalizing foreign exchange market protocols. They show that rigorous specification and verification of protocols via commitments solve many issues that emerge in existing systems due their informal business semantics. Singh, Chopra, and Desai (2009) propose a commitment-based service oriented architecture, which replaces invocation-based service objects with engagement-based autonomous business services. A set of transaction patterns over commitments are proposed, which reflect business requirements of various common business transactions, and provide basic building blocks to develop complex interactions. Benefits of this approach are flexible enactment of transactions, and ease of specification and composition of business processes. Kafalı, Günay, and Yolum (2012, 2013) develop a method using commitments to capture violation of privacy policies in social networks. They model privacy policies between parties in social networks using commitments, and employ model checking on these policies to capture violations. Their proposed approach can capture privacy breaches that cannot be detected in traditional systems. Furthermore, a prediction tool is also developed that utilizes Semantic Web technologies to capture potential privacy breaches that may occur in the future due to evolution of social network relations. Telang, Kalia and, Singh (2012, 2015) conducted an experimental evaluation of the commitment-based Comma methodology, and show that Comma outperforms the traditional HL7 Messaging Standard for healthcare process modeling. Chopra and Singh (2016b) introduce Interaction-Oriented Software Engineering as a commitment-based paradigm to capture social aspects of sociotechnical system emphasizing openness, autonomy, and accountability. Chopra and Singh (2016a) also develop Custard framework, which provides a relational schema and queries for commitments and their lifecycle, to build an abstraction layer over underlying information stores such as databases.

7. Discussion

In this paper we presented ProMoca framework for modeling and analyzing agent behaviors in commitment protocols taking uncertainty into account. Our main motivation for developing ProMoca is to analyze a target agent's behavioral properties when enacting a commitment protocol with respect to its goals, and beliefs about other agents' uncertain behaviors. In the recent years, commitments are applied to solve practical challenges in many domains such as e-commerce, sociotechnical systems, privacy, security, and healthcare. Modeling and verification are essential aspects of the development process for such practical systems, mainly to ensure correctness and effectiveness of a systems. ProMoca provides a novel analysis tool to handle these two key aspects of development by providing an expressive modeling language and an efficient model checking algorithm. The modeling language of ProMoca is expressive enough to model practical situations as we demonstrate in our examples. In fact, ProMoca provides more expressive power than needed by most of the previous work on commitments. The model checking algorithm of ProMoca is efficient and can handle complex situations as our empirical results show.

However, ProMoca has also certain limitations. In terms of modeling, ProMoca can be extended to model a more wider range of practical cases by introducing new language elements as we discuss in our future work at the end of this section. In terms of verification, well-known scalability issues of probabilistic model checking applies also to ProMoca. However, our results show ProMoca's efficiency in many practical situations. Our comparison with the state-of-the-art general purpose probabilistic model checker PRISM also shows that ProMoca outperforms PRISM when verifying an agent's compliance and goal satisfaction in a commitment protocol. Finally, social commitments framework itself is not an all-around solution to model every practical situation. However, it can be integrated with other approaches in multiagent systems to model and reason about more complex practical situations, e.g., integration with artifacts (Baldoni, Baroglio, & Capuzzimati, 2015) and other normative concepts such as prohibitions and authorizations (Chopra & Singh, 2016a).

As we stated earlier, there are various general purpose tools, such as PRISM and MCMAS, which can be used to verify agents with respect to commitment protocols. There are also several reasoning methodologies to handle uncertainty (Eiter & Lukasiewicz, 2003; Richardson & Domingos, 2006). Let us justify, why we develop a new tool while such tools and methodologies already exist. There are mainly two motivations behind the development of ProMoca as a new tool. The first one is the ease of modeling. Since general purpose tools do not support commitments, they do not provide any facilities to model them. Therefore, to be able to use one of these tools for commitments, first a model of a commitment should be developed in the tool. According to our experience, this is a non-trivial and error-prone task. Furthermore, such models are mostly developed considering some specific properties that are aimed to be verified in a target system. Hence, reuse of such models for other properties and systems is also limited. ProMoca solves these issues by providing an expressive modeling language that includes various facilities to model commitments. Hence, users can easily model commitment protocols, without worrying about the underlying model of commitments. Besides, ProMoca is based on a general model of commitments that is independent from particular properties and application specific assumptions. Hence, it can be used in any domain to verify arbitrary properties. Our second motivation is efficiency. As our experimental results clearly demonstrate, the state-of-the-art probabilistic model checker PRISM cannot achieve efficiency of ProMoca while verifying an agent's compliance and goal satisfaction in a commitment protocol. The main reason of ProMoca's efficiency is the utilization of commitment semantics to efficiently verify the addressed properties.

ProMoca can be used to model a wide majority of commitment protocols that are considered in the previous work. However, it is still open for many improvements. A major improvement is to extend ProMoca with an explicit representation of time. Currently, time can be modeled in ProMoca in an abstract manner using regular variables as we demonstrated in our examples. This is sufficient for many domains, however, especially for modeling commitments and agent in real-time systems, an explicit notion of time is necessary. Another improvement is introduction of numerical variables and arithmetic operations to ProMoca. Such variables are necessary to precisely model resource related issues (e.g., money, number of available spare parts, etc.). Addition of these features is fairly straightforward from syntactic and semantic point of view. However, verification of time and numerical variables increase complexity of model checking substantially. Hence, development of novel abstraction and reduction techniques that use commitment semantics, is essential for efficient and scalable model checking of such models. ProMoca can also be extended with more syntactic elements to simplify modeling of commitments. An example is the use of parameters in commitment specifications for modeling generic commitments.

Beside the above improvements to ProMoca, we also aim to extend ProMoca to model other commitment concepts such as choice and coordination (Baldoni, Baroglio, Chopra, Desai, Patti, & Singh, 2009), and relevant properties such as feasibility (Günay &

Yolum, 2013) and safety (Marengo et al., 2011). Besides, we plan to support other normative concepts such as obligations and prohibitions, which are relevant to commitments (Boella & van der Torre, 2004; Craven & Sergot, 2008; Ågotnes, van der Hoek, & Wooldridge, 2010; Criado, Argente, & Botti, 2011). Last but not least integration of ProMoca with the recent work on dynamic protocol creation in open systems is an interesting future work (Yolum & Singh, 2007; Artikis, 2009; Meneguzzi, Telang, & Singh, 2013; Günay, Winikoff, & Yolum, 2013, 2015; Cranefield, Savarimuthu, Meneguzzi, & Oren, 2015). This research aims to automate creation of protocols at run-time, which requires agents to agree on a commitment protocol to regulate their interaction according to their own requirements. To achieve this, individual agents should be able of analyze their behaviors with respect to their requirements and a commitment protocol, where ProMoca can be a valuable tool.

Acknowledgments

We thank to the anonymous reviewers for their insightful comments. This work is supported by "Formal Verification on Cloud" project under Grant No: M4081155.020 and "Bring the Advanced Model Checking Techniques to Real-world Problems" project under Grant No: M4011178.020.

References

Alberti, M., Chesani, F., Gavanelli, M., Lamma, E., Mello, P., & Torroni, P. (2008). Verifiable agent interaction in abductive logic programming: The $\mathcal{S}$CIFF framework. *ACM Transactions on Computational Logic*, *9*(4), 29:1–29:43.

Aldewereld, H., Vázquez-Salceda, J., Dignum, F., & Meyer, J.-J. C. (2006). Verifying norm compliancy of protocols. In *Agents, Norms and Institutions for Regulated Multi-Agent Systems*, pp. 231–245.

Alur, R., Henzinger, T. A., & Kupferman, O. (2002). Alternating-time temporal logic. *Journal of the ACM*, *49*(5), 672–713.

Andrés, M. E., D'Argenio, P., & Rossum, P. (2009). Significant diagnostic counterexamples in probabilistic model checking. In *Proceedings of the 4th International Haifa Verification Conference on Hardware and Software: Verification and Testing*, pp. 129–148. Springer-Verlag.

Artikis, A. (2009). Dynamic protocols for open agent systems. In *Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems*, pp. 97–104.

Baier, C., & Katoen, J.-P. (2008). *Principles of Model Checking.* The MIT Press.

Baldoni, M., Baroglio, C., & Capuzzimati, F. (2015). Programming JADE and Jason agents based on social relationships using a uniform approach. In Koch, F., Guttmann, C., & Busquets, D. (Eds.), *Advances in Social Computing and Multiagent Systems*, Vol. 541, pp. 167–184. Springer.

Baldoni, M., Baroglio, C., Chopra, A. K., Desai, N., Patti, V., & Singh, M. P. (2009). Choice, interoperability, and conformance in interaction protocols and service choreographies. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems*, pp. 843–850.

Bellman, R. (1957). Markovian decision processes. *Journal of Mathematics and Mechanics*, *38*, 716–719.

Bhat, G., Cleaveland, R., & Groce, A. (2001). Efficient model checking via Büchi tableau automata. In *Proceedings of the 13th International Conference on Computer Aided Verification*, pp. 38–52.

Boella, G., & van der Torre, L. (2004). Regulative and constitutive norms in normative multiagent systems. In *Proceedings of 9th International Conference on the Principles of Knowledge Representation and Reasoning*, pp. 255–265.

Bryant, R. E. (1986). Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Compututers*, *35*(8), 677–691.

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

Chittaro, L., & Montanari, A. (1996). Efficient temporal reasoning in the cached event calculus. *Computational Intelligence*, *12*(3), 359–382.

Chopra, A. K., Dalpiaz, F., Giorgini, P., & Mylopoulos, J. (2010). Reasoning about agents and protocols via goals and commitments. In *Proceedings of the Ninth International Conference on Autonomous Agents and Multiagent Systems*, pp. 457–464.

Chopra, A. K., & Singh, M. P. (2015). Cupid: Commitments in relational algebra. In *Proceedings of 29th AAAI Conference on Artificial Intelligence*, pp. 2052–2059.

Chopra, A. K., & Singh, M. P. (2016a). Custard: Computing norm states over information stores. In *Proceedings of the 2016 International Conference on Autonomous Agents and Multiagent Systems*, pp. 1096–1105.

Chopra, A. K., & Singh, M. P. (2016b). From social machines to social protocols: Software engineering foundations for sociotechnical systems. In *Proceedings of the 25th International Conference on World Wide Web*, pp. 903–914.

Clarke, Jr., E. M., Grumberg, O., & Peled, D. A. (1999). *Model Checking*. MIT Press, Cambridge, MA, USA.

Cranefield, S., Savarimuthu, T., Meneguzzi, F., & Oren, N. (2015). A bayesian approach to norm identification. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, pp. 1743–1744.

Craven, R., & Sergot, M. (2008). Agent strands in the action language n$\mathcal{C}$+. *Journal of Applied Logic*, *6*(2), 172–191.

Criado, N., Argente, E., & Botti, V. (2011). Open issues for normative multi-agent systems. *AI Communications*, *24*(3), 233–264.

Desai, N., Cheng, Z., Chopra, A. K., & Singh, M. P. (2007a). Toward verification of commitment protocols and their compositions. In *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems*, pp. 33:1–33:3.

Desai, N., Chopra, A. K., Arrott, M., Specht, B., & Singh, M. P. (2007b). Engineering foreign exchange processes via commitment protocols. In *IEEE International Conference on Services Computing*, pp. 514–521.

Desai, N., Chopra, A. K., & Singh, M. P. (2009). Amoeba: A methodology for modeling and evolving cross-organizational business processes. *ACM Transactions on Software Engineering and Methodology*, *19*(2), 6:1–6:45.

Desai, N., Narendra, N. C., & Singh, M. P. (2008). Checking correctness of business contracts via commitments. In *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems*, pp. 787–794.

Eiter, T., & Lukasiewicz, T. (2003). Probabilistic reasoning about actions in nonmonotonic causal theories. In *Proceedings of the Nineteenth Conference on Uncertainty in Artificial Intelligence*, pp. 192–199.

El Kholy, W., Bentahar, J., Menshawy, M. E., Qu, H., & Dssouli, R. (2014). Conditional commitments: Reasoning and model checking. *ACM Transactions on Software Engineering Methodology*, *24*(2), 9:1–9:49.

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

Fagin, R., Halpern, J. Y., Moses, Y., & Vardi, M. Y. (2003). *Reasoning About Knowledge*. MIT Press, Cambridge, MA, USA.

Fornara, N., & Colombetti, M. (2002). Operational specification of a commitment-based agent communication language. In *Proceedings of the 1st International Joint Conference on Autonomous Agents and Multiagent Systems*, pp. 536–542.

Gerard, S. N., & Singh, M. P. (2013). Formalizing and verifying protocol refinements. *ACM Transactions on Intelligent Syststems and Technology*, *4*(2), 21:1–21:27.

Giunchiglia, E., Lee, J., Lifschitz, V., McCain, N., & Turner, H. (2004). Nonmonotonic causal theories. *Artificial Intelligence*, *153*(1-2), 49–104.

Günay, A., Songzheng, S., Liu, Y., & Zhang, J. (2015). Automated analysis of commitment protocols using probabilistic model checking. In *Proceedings of 29th AAAI Conference on Artificial Intelligence*, pp. 2060–2066.

Günay, A., Winikoff, M., & Yolum, P. (2013). Commitment protocol generation. In *Declarative Agent Languages and Technologies X*, Vol. 7784 of *LNAI*, pp. 136–152. Springer.

Günay, A., Winikoff, M., & Yolum, P. (2015). Dynamically generated commitment protocols in open systems. *Journal of Autonomous Agents and Multi-agent Systems*, *29*, 192–229.

Günay, A., & Yolum, P. (2011). Detecting conflicts in commitments. In Sakama, C., Sardina, S., Vasconcelos, W., & Winikoff, M. (Eds.), *Declarative Agent Languages and Technologies IX*, Vol. 7169 of *LNAI*, pp. 51–66. Springer.

Günay, A., & Yolum, P. (2013). Constraint satisfaction as a tool for modeling and checking feasibility of multiagent commitments. *Applied Intelligence*, *39*(3), 489–509.

Halpern, J. Y. (2003). *Reasoning About Uncertainty.* MIT Press, Cambridge, MA, USA.

Han, T., Katoen, J.-P., & Berteun, D. (2009). Counterexample generation in probabilistic model checking. *IEEE Transactions on Software Engineering*, *35*(2), 241–257.

Hoare, C. A. R. (1978). Communicating sequential processes. *Communications of ACM*, *21*(8), 666–677.

Holzmann, G. J. (Ed.). (2004). *The SPIN Model Checker: Primer and Reference Manual.* Addison-Wesley.

Jakob, M., Pěchouček, M., Miles, S., & Luck, M. (2008). Case studies for contract-based systems. In *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems: Industrial Track*, pp. 55–62.

Kafalı, Ö., Günay, A., & Yolum, P. (2012). $\mathcal{PROTOSS}$: A run time tool for detecting $\mathcal{PR}$ivacy vi$\mathcal{O}$la$\mathcal{T}$ions in $\mathcal{O}$nline $\mathcal{S}$ocial network$\mathcal{S}$. In *IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, pp. 429–433.

Kafalı, Ö., Günay, A., & Yolum, P. (2013). Detecting and predicting privacy violations in online social networks with $\mathcal{PROTOSS}$. *Distributed and Parallel Databases*, *32*(1), 161–190.

Kafalı, Ö., Günay, A., & Yolum, P. (2014). GOSU: Computing goal support with commitments in multiagent systems. In *Proceedings of 21st European Conference on Artificial Intelligence*, pp. 477–482.

Kafalı, Ö., & Torroni, P. (2012). Exception diagnosis in multiagent contract executions. *Annals of Mathematics and Artificial Intelligence*, *64*(1), 73–107.

Kwiatkowska, M., Norman, G., & Parker, D. (2011). PRISM 4.0: Verification of probabilistic real-time systems. In *Proceedings of the 23rd International Conference on Computer Aided Verification*, pp. 585–591.

Lomuscio, A., Qu, H., & Raimondi, F. (2009). MCMAS: A model checker for the verification of multi-agent systems. In *Proceedings of the 21st International Conference on Computer Aided Verification*, pp. 682–688.

Mallya, A. U., & Huhns, M. N. (2003). Commitments among agents. *IEEE Internet Computing*, *7*(4), 90–93.

Marengo, E., Baldoni, M., Baroglio, C., Chopra, A. K., Patti, V., & Singh, M. P. (2011). Commitments with regulations: Reasoning about safety and control in REGULA. In *Proceedings of the Tenth International Conference on Autonomous Agents and Multiagent Systems*, pp. 467–474.

Meneguzzi, F., Telang, P. R., & Singh, M. P. (2013). A first-order formalization of commitments and goals for planning. In *Proceedings of the 27th AAAI Conference on Artificial Intelligence*, pp. 697–703.

Modgil, S., Faci, N., Meneguzzi, F., Oren, N., Miles, S., & Luck, M. (2009). A framework for monitoring agent-based normative systems. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems*, pp. 153–160.

Montali, M., Calvanese, D., & De Giacomo, G. (2014). Verification of data-aware commitment-based multiagent system. In *Proceedings of the 2014 International Conference on Autonomous Agents and Multi-agent Systems*, pp. 157–164.

Pnueli, A. (1977). The temporal logic of programs. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science*, pp. 46–57.

Ågotnes, T., van der Hoek, W., & Wooldridge, M. (2010). Robust normative systems and a logic of norm compliance. *Logic Journal of IGPL*, *18*(1), 4–30.

Richardson, M., & Domingos, P. (2006). Markov logic networks. *Machine Learning*, *62*(1-2), 107–136.

Schmalz, M., Varacca, D., & Völzer, H. (2009). Counterexamples in probabilistic ltl model checking for markov chains. In *Proceedings of the 20th International Conference on Concurrency Theory*, pp. 587–602. Springer-Verlag.

Singh, M. P. (1999). An ontology for commitments in multiagent systems: Toward a unification of normative concepts. *Artificial Intelligence and Law*, *7*(1), 97–113.

Singh, M. P. (2008). Semantical considerations on dialectical and practical commitments. In *Proceedings of the 23rd National Conference on Artificial Intelligence*, pp. 176–181.

Singh, M. P., Chopra, A. K., & Desai, N. (2009). Commitment-based service-oriented architecture. *IEEE Computer*, *42*(11), 72–79.

Sirbu, M. A., & Tygar, J. D. (1995). NetBill: An internet commerce system optimized for network delivered services. *IEEE Personal Communications*, *2*(4), 34–39.

Sultan, K., Bentahar, J., Wan, W., & Al-Saqqar, F. (2014). Modeling and verifying probabilistic multi-agent systems using knowledge and social commitments. *Expert Systems and Applications*, *41*(14), 6291–6304.

Sun, J., Liu, Y., Dong, J. S., & Pang, J. (2009). PAT: Towards flexible verification under fairness. In *Proceedings of the 21th International Conference on Computer Aided Verification (CAV)*, Vol. 5643 of *Lecture Notes in Computer Science*, pp. 709–714. Springer.

Telang, P., & Singh, M. (2012). Specifying and verifying cross-organizational business models: an agent-oriented approach. *IEEE Transations on Services Computing*, *5*(3), 305–318.

Telang, P. R., Kalia, A. K., & Singh, M. P. (2015). Modeling healthcare processes using commitments: An empirical evaluation. *PLoS ONE*, *10*(11), doi:10.1371/journal.pone.0141202.

Telang, P. R., & Singh, M. P. (2012). Comma: A commitment-based business modeling methodology and its empirical evaluation. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems*, pp. 1073–1080.

Torroni, P., Chesani, F., Mello, P., & Montali, M. (2010). Social commitments in time: Satisfied or compensated. In *Proceedings of the 7th International Conference on Declarative Agent Languages and Technologies*, pp. 228–243, Berlin, Heidelberg. Springer-Verlag.

van Riemsdijk, M. B., Dastani, M., & Meyer, J.-J. C. (2009). Goals in conflict: Semantic foundations of goals in agent programming. *Autonomous Agents and Multi-Agent Systems*, *18*(3), 471–500.

Vasconcelos, W. W. (2005). Norm verification and analysis of electronic institutions. In *Proceedings of the Second International Conference on Declarative Agent Languages and Technologies*, pp. 166–182.

Yolum, P. (2007). Design time analysis of multiagent protocols. *Data and Knowledge Engineering*, *63*(1), 137–154.

Yolum, P., & Singh, M. P. (2002). Flexible protocol specification and execution: Applying event calculus planning using commitments. In *Proceedings of the 1st International Joint Conference on Autonomous Agents and Multiagent Systems*, pp. 527–534.

Yolum, P., & Singh, M. P. (2007). Enacting protocols by commitment concession. In *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems*, pp. 27:1–27:8.