Probabilistic and Epistemic Model Checking for Multi-Agent Systems

Wei Wan

A Thesis

In

The Department

Of

Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of Doctor of Philosophy at

Concordia University

Montréal, Québec, Canada

October 2014

Concordia University

School of Graduate Studies

This is to certify that the thesis prepared

By: **Ms. Wei Wan**

Entitled: **Probabilistic and Epistemic Model Checking for Multi-Agent Systems**

and submitted in partial fulfilment of the requirements for the degree of

Doctor of Philosophy (Computer Engineering)

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

_____ Dr. TBD (Chair)

_____ Dr. Nadia Tawbi

_____ Dr. Olga Ormandjieva

_____ Dr. Roch Glitho

_____ Dr. Abdelwahab Hamou-Lhadj

_____ Dr. Jamal Bentahar

_____ Dr. Abdessamad Ben Hamza

Approved by _____

Chair of the ECE Department

_____ 2014 _____

Dean of Engineering

Abstract

Probabilistic and Epistemic Model Checking for Multi-Agent Systems

Wei Wan
Doctor of Philosophy (Computer Engineering)
Concordia University, 2014

Model checking is a formal technique widely used to verify security and communication protocols in epistemic multi-agent systems against given properties. Qualitative properties such as safety and liveliness have been widely analyzed in the literature. However, systems also have quantitative and uncertain (i.e., probabilistic) properties such as degree of reliability and reachability, which still need further attention from the model checking perspective. In this dissertation, we analyze such properties and present a new method for probabilistic model checking of epistemic multi-agent systems specified by a new probabilistic-epistemic logic PCTLK. We model multi-agent systems distributed knowledge bases using probabilistic interpreted systems. We also define transformations from those interpreted systems into discrete-time Markov chains and from PCTLK formulae to PCTL formulae, an existing extension of CTL with probabilities. By so doing, we are able to convert the PCTLK model checking problem into the PCTL one. We address the problem of verifying probabilistic properties and epistemic properties in concurrent probabilistic systems as well. We then prove that model checking a formula of PCTLK in concurrent probabilistic systems is PSPACE-complete. Furthermore, we represent models associated with PCTLK logic symbolically with Multi-Terminal Binary Decision Diagrams (MTBDDs).

Finally, we make use of PRISM, the model checker of PCTL without adding new computation cost. Dining cryptographers protocol is implemented to show the applicability of the proposed technique along with performance analysis and comparison in terms of execution time and state space scalability with MCK, an existing epistemic-probabilistic model checker, and MCMAS, a model checker for multi-agent systems. Another example, NetBill protocol, is also implemented with PRISM to verify probabilistic epistemic properties and to evaluate the complexity of this verification.

To My Husband and all My Family Members

Acknowledgments

I take this opportunity to express my profound gratitude and deep regard to my supervisors *Dr. Jamal Bentahar* and *Dr. Abdessamad Ben Hamza* for their exemplary guidance, monitoring, constant encouragement, enduring patience, and financial support throughout the course of this thesis work. Everything that I have learned from them shall carry me a long way in my journey of life into which I am about to embark. I am grateful for the opportunity to study in Electrical and Computer Engineering (ECE), which has provided an excellent environment in which to cross-fertilize research ideas.

I thank all my co-authors, discussion partners and all those who have commented on the papers that I have written as a basis for this work. I would like to thank all of the people in my laboratory for their discussions on knowledge representation, model checking techniques, logics, and programming. I would also like to thank all the faculty and staff at ECE and CIISE for their assistance during my studies.

I am most fortunate to have loving family members. I dedicate this thesis to them for their enthusiasm and expectations of me. Thank you *Bad, Mom, my sister Hong, my father-in-law Dr. Stan Shapiro, my brother Jun, Pam, Jun Wang* . Sincere appreciation is extended to my mother-in-law, *Dr. Anna-Beth Doyle* for her immense help to me in editing my papers and thesis. Special thanks is extended to my beloved husband *Mark* for helping me face my difficulties, improve my vocabulary, and for his enduring support, and endless love.

Special thanks to my friends (in alphabetical order), without them, my student life would not have been so much fun: *Dongbai, Helen, Keichun, Li, Ming, Peijun, Qingwei, Tao, Wei Wu*.

Last but not least, I would like to thank *Fonds de recherche nature et technologies* (FQRNT) for its financial support which has helped me sustain my research.

Table of Contents

List of Tables

List of Figures

List of Acronyms

ARCTL	Action-Restricted Computation Tree Logic
BDD	Binary Decision Diagram
BDI	Belief-Desire-Intention
CDC	Cheating Dining Cryptographers Protcol
CTL	Computation Tree Logic
CTLK	Computation Tree Logic for Knowledge
CTMC	Continues-Time Markov Chain
CSL	Continuous Stochastic Logic
DC	Dining Cryptographers Protocol
DTMC	Discrete-Time Markov Chain
LTL	Linear Temporal Logic
MC	Markov Chain
MDP	Markov Decision Process
MTBDD	Multi-Terminal Binary Decision Diagram
OBDD	Ordered Binary Decision Diagram
PCTL	Probabilistic Computation Tree Logic
PCTLK	Probabilistic Computation Tree Logic of Knowledge

Chapter 1

Introduction

In this chapter, we explain what initiated our interest in probabilistic and epistemic model checking in Multi-agent systems (MAS). We also specify research problems under consideration, propose solutions, and present the structure of this dissertation.

1.1 Research Scope

Agents are autonomous entities with reactive, pro-active, and rational properties. MAS are comprised of a set of intelligent agents interacting with each other to solve problems that are difficult or impossible for an individual agent. In the past two decades, MAS have been very successfully implemented as a new paradigm for Web services, network security, and distributed systems, amongst others. However, MAS are complex systems where functional and non-functional properties cannot be checked easily by simply inspecting the system models. Since it is very expensive to modify MAS after deployment, it is desirable to have automatic verification methods apriori checking the correctness of designed systems. Model checking [32] is one of the most widely used automatic techniques to verify whether or not system design models satisfy given requirements. In model checking, the system to be verified is described as a

model of a given logic, while the property to be verified is expressed as a formula in the same logic. Model checking is a well-designed formal technique allowing the automatic verification of the models against specific properties that capture the requirements. Many researchers have proposed a variety of model checking approaches for modeling and verifying MAS [8, 10, 22, 39, 47, 57, 59, 78, 79, 95, 99]. Three processes for model checking these systems, namely modeling, specification, and verification, are the main foci in this dissertation.

Models, i.e., formal description and representation of the structure, behavior, and object of systems, are used to help us know and understand the subject matter that they represent. A Kripke structure, one of the most popular models in model checking, is a finite automaton that represents reachable states and state transitions in a graph. There are many different formalizations of Kripke structures. Interpreted systems, one type of formalization of a Kripke structure, is frequently used for defining the semantics to reason about time and knowledge in MAS. However, a finite automaton cannot express likelihood activities. If there are uncertainties about the system's events, a probabilistic counterpart of finite automata, Markov chains, are used to represent the system models. Markov chains are transition systems with probabilistic distributions over the transitions. In our research, we focus on interpreted systems and Markov chains for representing epistemic (i.e., about knowledge) and probabilistic models.

Specification, a formal method of expression the requirements of a system, is usually given in some logical formalism. For software systems, specification often uses temporal logics to assert how the behavior of the system evolves over time. Temporal logics are often classified into linear or branching according to the underlying time structure. Linear Temporal Logic (LTL) is a basic linear structure logic while Computation Tree Logic (CTL) [44] is a basic branching structure. Several specific

languages are extensions of the basic propositional language to express epistemic, probabilistic, and dynamic properties. Examples of those languages are Computation Tree Logic for Knowledge (CTLK) [72], Probabilistic Computation Tree Logic (PCTL) [52], Probabilistic Branching Time Logic (PBTL) [7], Continuous Stochastic Logic (CSL), etc. In this dissertation, we design a new language that can express both probabilistic and epistemic properties.

Formal verification refers to the algorithms used to verify whether or not system models satisfy specification properties. For verifying specific properties such as epistemic and probabilistic properties, we can either develop dedicated tools, for example MCK (Model Checking Knowledge) [47] and MCMAS (Model Checking MAS) [78] for epistemic properties, PRISM (Probabilistic Model Checking) [67] for probabilistic ones, or refine some existing model checking tools to suit the specific properties. Examples of proposals using the second approach include [105] where the authors proposed an imperative programming language, MABLE, to specify MAS along with a Belief-Desire-Intention (BDI) logic to express the properties and showed how to verify this language using SPIN [55]. Similarly, Lomuscio et al. [73] described a reduction process from model checking CTLK into the problem of verifying Action-Restricted CTL (ARCTL). The authors extended the NuSMV model checker to verify epistemic properties in CTLK. In our research, we focus on refinement of existing model checking tools by specifying general construct models to represent specific systems. Therefore, a reduction model checking algorithm is developed for verifying epistemic and probabilistic properties. Complexity analysis of this algorithm is also included in this dissertation.

1.2 Motivations

Epistemic logic [82] provides a formal language for reasoning about states of knowledge in Artificial Intelligence (AI), especially in MAS. Epistemic logic is an instance of multimodal logic that includes temporal modalities and epistemic modalities. The basic epistemic modality is K, which is used to express "agent a knows that...", written as $K_a\phi$, where ϕ is the content that agent knows. This formula is true if and only if in all agent a's possible worlds, the content ϕ is true. Indeed the epistemic modal logic is considered as $S5_n$ combined with a temporal logic [104] interpreted on interpreted systems [45]. Epistemic modalities have also been investigated within a first order logic to reason about knowledge and time in a first order setting [8].

In conventional model checking, the results of verification only focus on the absolute accuracy of properties in the model being constructed. For example, we can verify "it is always possible to restart the system after failure", or in a card game scenario that "agent 1 knows for sure that agent 2 has a diamond Ace card". However, there is uncertain and/or incomplete knowledge to be expressed as well. For instance, in distributed systems, situations such as "the message will be delivered successfully with probability of 95%", or in auction systems that "agent a knows that he only has 70% chance to get the auction item", or in a card game scenario that "agent b only knows 80% of the information". To express these uncertainties, we need logics that can express quantitative aspects. Knowledge is extremely important for an agent to make decisions, and these decisions strongly depend on the confidence the agent has in his knowledge. Knowing a fact for sure and reacting according to it are definitely different from reacting to the same fact knowing that it holds with 50% probability. For illustration, if the agent knows that the message will be successfully delivered with a probability 0.5, then she will probably consider other solutions such as sending duplicate copies in order to increase the chance of successful delivery. Accounting

for stochastic phenomena of epistemic systems and verifying the correctness of such systems in uncertain environments are key aspects in concrete applications [39, 59]. It is also important to analyze *quantitative properties*, such as reliability, responsiveness, degree of reachability, and degree of persistence (does eventually an event always hold?). These quantitative properties and uncertainty cannot be expressed in regular epistemic logics. Therefore, we need to consider probabilistic aspects when modeling the system to allow providing such estimations by assessing the likelihood of different events.

Probabilistic logic [52] aims to combine the capacity of probability theory to handle uncertainty with the capacity of logic to express and reason about facts. Several probabilistic logics [37, 7, 13, 24, 54, 67] have been proposed to specify probabilistic properties, such as "the system satisfies property ϕ with probability at least p". Like epistemic logic, probabilistic logic is an instance of multimodal logic. Probabilistic logic usually includes temporal modalities and probabilistic modalities. Probabilistic modalities vary among logics. A probabilistic operator $\mathcal{P}_{\bowtie p}$ is adopted by many probabilistic logics, where $\bowtie \in \{\leq, <, >, \geq\}$ and $p \in [0, 1]$. A basic formula for probabilistic logic is of the form $\mathcal{P}_{>0.95}\phi$. This formula is true at a given state s if and only if the probability of all the infinite paths starting from that state and satisfying ϕ is greater than 0.95. (for basic notions of measure and probability theory, see [40]). Nevertheless, as mentioned in previous examples, probabilistic logic cannot express uncertain/incomplete knowledge. We will explore methods to express uncertain knowledge in this dissertation.

Using interpreted systems to formalize agents' models has been useful in representing, modeling, and verifying epistemic systems [41, 56, 78, 95, 83, 85]. Interpreted

systems capture the philosophical foundations of knowledge naturally by using possible and accessible worlds, agent local states, and system global states. In this formalism, an agent obtains his knowledge from the information stored in all equivalent states of the current local state, which means that the agent cannot distinguish those states. Thus, an agent knows ϕ in a given global state if and only if ϕ is true in all its global accessible states, in which the local components of that particular agent are equivalence. Nevertheless, the specification of agents' uncertainty of knowledge with interpreted systems is still in the early stages and needs to be further investigated. In this dissertation, we introduce the new formalism of probabilistic interpreted systems and use it to express not only certain, but also quantitative and uncertain knowledge.

As a model grows, model checking can face a serious state explosion problem, which means that the size of the model grows exponentially with the number of components. One technique to alleviate this problem is symbolic model checking on Ordered Binary Decision Diagrams (OBDDs). However, OBDDs cannot represent probabilistic systems. Recently, Multi-Terminal Decision Diagrams (MTBDDs) [30] have been used successfully to represent probabilistic systems and have been applied in several probabilistic model checking approaches [37, 26, 70, 103]. MTBDDs have the same structure as BDDs, except that terminals in MTBDDs can be real numbers other than 0 and 1. MTBDDs are known to be compact and efficient representations for sparse matrices [30]. We will explore the issue of representing probabilistic interpreted systems by MTBDDs data structure in our research in order to apply symbolic model checking techniques to probabilistic and epistemic logic.

1.3 Research Questions

As described above, it is necessary to verify uncertain epistemic properties in MAS. There are two questions that must be answered in order to check quantitative-epistemic properties: how to specify measurable epistemic properties and how to represent models capturing measurable epistemic features? Quantitative knowledge can be represented using probabilities and MAS can be modeled as a probabilistic Kripke-like model. Nevertheless, neither interpreted systems that formalize agents' models nor Markov chains that describe probabilistic models are able to express uncertain MAS completely and accurately. The problem of state explosion that exists for non-probabilistic systems remains a problem when we specify the requirements of uncertain MAS. In this dissertation, we 1) build a model that captures uncertainty in MAS; 2) create a new language to express uncertain epistemic properties; and 3) develop a new technique to verify whether or not probabilistic models satisfy probabilistic-epistemic properties.

To verify epistemic and probabilistic properties in MAS, the issues we face include:

1. How can we model uncertainty of the knowledge in MAS in order to formalize probabilistic (or quantitative) and epistemic properties for future model verification?

2. How can we specify probabilistic MAS to formulate systems requirements?

3. What model checking techniques and algorithms can be used to verify epistemic and probabilistic properties?

4. How can we refine existing model checking tools to verify epistemic and probabilistic models?

5. Is the proposed approach decidable? What is the time and space complexity of this method?

1.4 Proposed Solutions

In this dissertation, we integrate Markov chains structure into interpreted systems to express probabilistic MAS. To specify the quantitative properties of these systems, a probabilistic-epistemic logic which combines the temporal logic with epistemic logic is defined at the probabilistic level by adding the degree of epistemic properties. Specifically, uncertain knowledge can be represented using probabilities and the MAS can be modeled as a probabilistic Kripke-like model. We extend interpreted systems for Discrete-Time Markov Chains (DTMC) M_{PIS} to specify probabilistic MAS. A new logic that can specify probabilistic and epistemic properties, called Probabilistic Computation Tree Logic for Knowledge (PCTLK), is introduced. The semantics for this logic is associated with the model M_{PIS}. Then, we design reduction-based model checking techniques for probabilistic MAS on PCTLK.

Our proposed research solutions are listed as follows:

1. Extend the conventional formalism of interpreted systems by allowing transitions with probabilities to describe probabilistic MAS. The extended interpreted systems are Kripke structure models for n agents with probability transitions.

2. Define probabilistic-epistemic temporal logic PCTLK. PCTLK not only allows probabilistic paths to be accounted for, but also represents uncertainty/quantified knowledge.

3. Develop model checking techniques for PCTLK. We define equivalence transformations from PCTLK formulae to PCTL formulae to convert the problem of model checking PCTLK to PCTL.

4. Analyze the complexity of PCTLK model checking. We prove that PCTLK model checking problem in concurrent probabilistic systems is PSPACE-complete and can be solved in polynomial time in the size of the model and length of the formula.

5. Represent probabilistic interpreted systems with MTBDDs.

1.5 Methodology

In this dissertation, we put forward probabilistic model checking to verify agents' uncertain knowledge in MAS. A general overview of our methodology is depicted in Figure 1.1. To apply our probabilistic model checking for MAS, the first step is to construct the formal system model and formal specifications. We need to model MAS in order to describe the behavior and knowledge of agents in a concise and unambiguous way; and define probabilistic epistemic model M_{PIS}. Properties should also be described in a precise and unambiguous manner using probabilistic and epistemic logic PCTLK. Then the model and properties are transformed into equivalent discrete time Markov chains and probabilistic computation tree logic PCTL. The second step is to run the PRISM model checker to check the validity of the properties in the system model. The ultimate objective is to analyze the model checking results. There are then three possible outcomes: positive, negative, or lack of memory.

In the case of positive result, we conclude that the property is satisfied in the system model. If a negative result is given, either we need to generate a counterexample and simulate to locate errors; or if the verified property is measurable, we can then calculate the probability and analyze the uncertainty by comparing the probability with the required value to locate errors. Then, we can refine the model design and repeat the verification process until the property is getting satisfied. If the model is

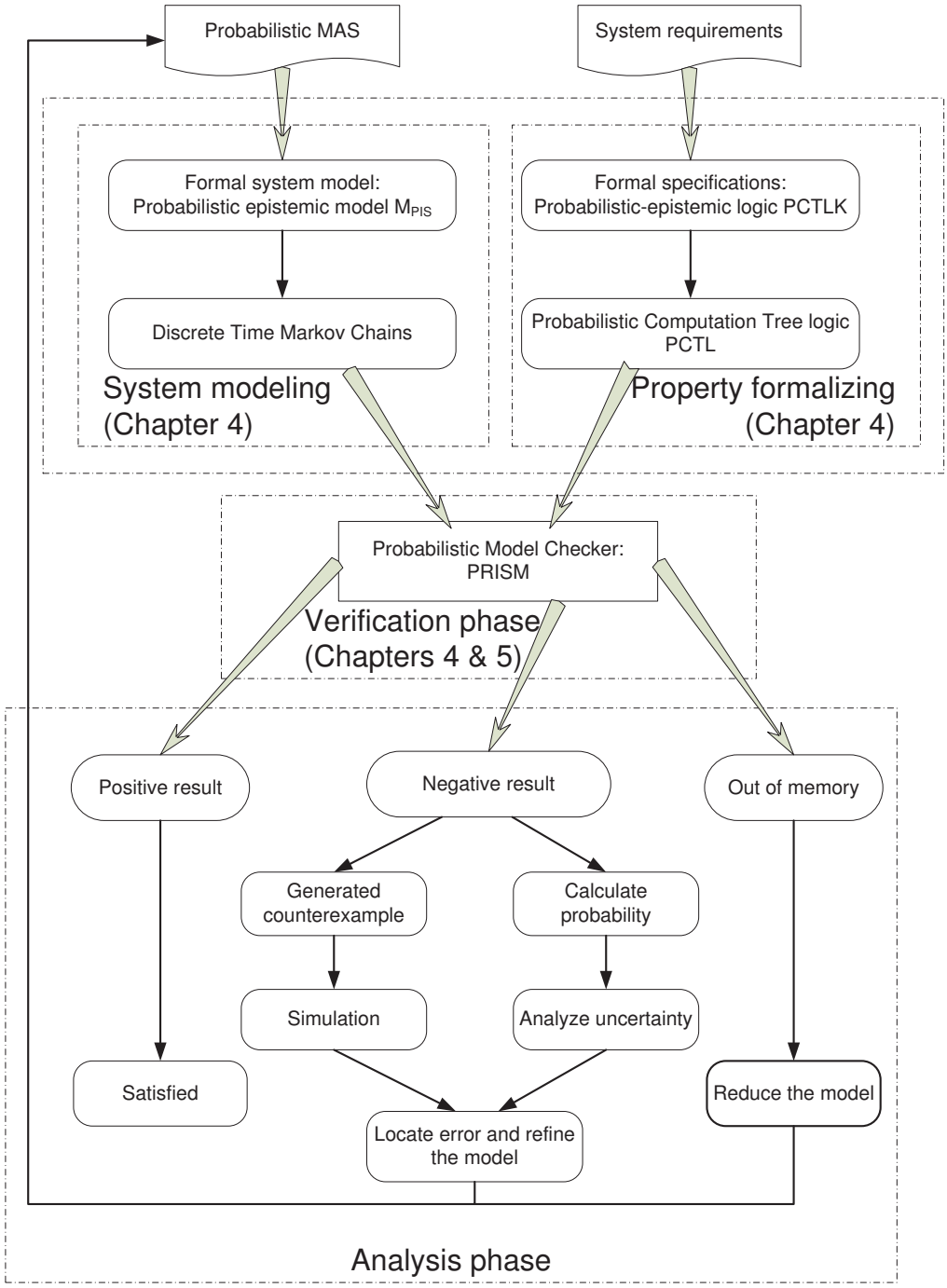

Figure 1.1: Methodology Overview

too large to be verified with the model checker, we need to consider some techniques that can reduce the system model and repeat the verification process until the size of the model becomes manageable.

1.6 Layout of This Dissertation

This dissertation is organized as follows. In Chapter 2, we briefly introduce the relevant background and relevant literature. We show how to model probabilistic systems and epistemic systems. We describe three kinds of probabilistic models: discrete-time Markov chains, continuous-time Markov chains, and Markov decision process along with the formalism of interpreted systems. We also review different model checking techniques and tools. In Chapter 3, we discuss three areas of related work: models, logics, and model checking approaches.

Chapter 4 and Chapter 5 present the main outcome of our research project. Chapter 4 is based on a published paper in *Knowledge-based Systems* [102]. The research questions 1 to 4 are answered in this chapter. We explain how we extend the regular interpreted systems to express probabilistic models and introduce the new formalism of probabilistic interpreted systems. We define a new logic PCTLK and state its syntax and semantics. Furthermore we explain how model checking PCTLK can be reduced to model checking PCTL and implement our approach with PRISM [66].

Chapter 5 is based on a manuscript [98] submitted to *Journal of Computers and Mathematics with Applications*. The answer to research question 5 is given in this chapter. We review concurrent probabilistic systems and explain how model checking PCTLK can be reduced to model checking PBTL. The complexity of PCTLK model checking is analyzed as well. Time complexity over Markov chains and space

complexity over concurrent probabilistic systems are proved to be polynomial in the size of the model and length of the formula. We then explore MTBDD-based symbolic model representation. We implement our approach using the MTBDD engine of PRISM and analyze the experimental results.

Finally, Chapter 6 concludes the dissertation. We summarize the main contributions of our research. At the end of this chapter, we list potential directions for future work.

Chapter 2

Background and Literature Review

This chapter presents and discusses the relevant background of probabilistic and epistemic model logic in MAS. In Section 2.1, we introduce how Kripke-like Markov chains are used to specify the probabilistic model. Interpreted systems that are used to express temporal epistemic models are discussed in Section 2.2. Computation Tree Logic of Knowledge (CTLK) is presented in Section 2.3. We also discuss two probabilistic temporal logics PCTL and PBTL in Section 2.4. Then we overview model checking techniques in Section 2.5 and compare several of the most popular model checkers in probabilistic or epistemic logic in Section 2.6.

2.1 Probabilistic Models

Probability is used in the design and analysis of an agent to measure the likelihood that some specific events will occur. In order to model these uncertaint events, measurable features such as probabilities should enrich transition systems. There are several methods of systems modeling to express probabilistic attributes. One of the most popular operational probabilistic models is Markov chains [49], named after Russian mathematician Andrei Markov (1856 -1922). Markov chains are probabilistic

finite automata with probability distributions over transitions and are widely used in modeling probabilistic properties in MAS [38, 39]. There are two types of Markov chain models: Discrete-Time Markov chains (DTMC) and Continuous-Time Markov Chains (CTMC). In DTMC, a system is in a given state at each "step", with the state changing randomly between steps. Thus, the next state in DTMC is chosen stochastically or probabilistically. In [6], a DTMC model is defined as follow:

Definition 2.1. *Over a set of atomic propositions AP, a model of DTMC can be expressed as a tuple* $\{S,\ \mathbf{P}\ ,\ I_{init},\ L,\ AP\}$*, where:*

- S *is a nonempty and finite set of states.*
- $\mathbf{P} : S \times S \to [0, 1]$ *is the transition probability function, such that for every state* $s \in S$*, we have* $\sum_{s' \in S} P(s, s') = 1$*.*
- $I_{init} : S \to [0, 1]$ *is the initial distribution such that for all states* $s \in S$*,* $\sum_{s \in S} I_{init}(s) = 1$*.*
- $L : S \to 2^{AP}$ *is a state labeling function.*

For mathematics treatment purposes, the initial distribution I_{init} can be viewed as an ordered list of row vectors $(I_{init}(s))_{s \in S}$, in which the value of every row represents the initial probability from all states in the model. The transition probability function $\mathbf{P} : S \times S \to [0, 1]$ is represented by the matrix $(\mathbf{P}(s, t))_{s,t \in S}$. The probabilities of state s to its successors t are shown on the row of the matrix, while the probabilities of entering state s from state t are shown on the column of the matrix.

Figure 2.1 (a) is a graph example of a DTMC model. $S = \{s_0, s_1, s_2, s_3\}$ is the set of states. The atomic proposition set is given by $AP = \{r,\ f,\ s\}$. State labeling functions are: $L(s_0) = \{\varnothing\}$, $L(s_1) = \{r\}$, $L(s_2) = \{s\}$ and $L(s_3) = \{f\}$. The initial distribution I_{init} and the transition probability function $\mathbf{P}$ viewed as a 4×4 matrix are as follows:

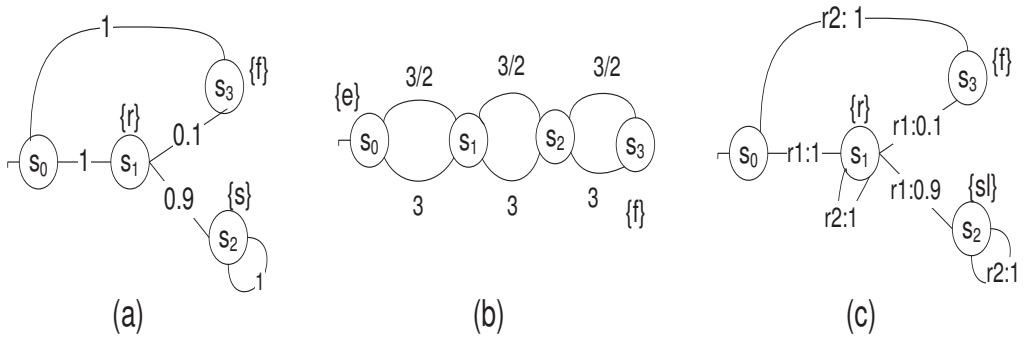

Figure 2.1: Markov Chains and Markov Decision Processes example

$$\mathbf{P} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0.9 & 0.1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \qquad I_{init} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

In DTMCs, the progress of time is modeled by discrete time steps, one for each transition of the model. However, the progress of time needs to be modeled as continue time in some systems. In CTMC, transition delays are assumed to be modeled by exponential distributions. According to [5], CTMC can be defined as follow:

Definition 2.2. *Over a set of atomic propositions AP, a model of CTMC can be expressed as a tuple* $\{S,\ s_{init},\ \mathbf{R},\ L,\ AP\}$*, where:*

- S, $L : S \to 2^{AP}$ *are as for DTMC*
- $s_{init} \in S$ *is the initial state.*
- $\mathbf{R} : S \times S \to \mathbb{R}^+$ *is the transition rate matrix.*

The matrix $\mathbf{R}$ assigns possible transitions for each pair of states if and only if $\mathbf{R}(s, s') > 0$ in CTMC models. A transition can occur if and only if it can be modeled as an exponential distribution with rate $\mathbf{R}(s, s')$ [68]. CTMC models are much more

complex than the DTMCs. As our main focus is on agents with discrete time steps, for the rest of this dissertation when we talk about Markov chain probabilistic models in MAS, we refer only to DTMC probabilistic models.

Figure 2.1 (b) is a graph example of a CTMC model. $S = \{s_0, s_1, s_2, s_3\}$ is the set of states. The initial state is s_0 and the atomic proposition is defined by $AP = \{e,\ f\}$. State labeling functions are: $L(s_0) = \{e\}$, $L(s_1) = L(s_2) = \{\varnothing\}$ and $L(s_3) = \{f\}$. The transition rate matrix $\mathbf{R}$ is:

$$\mathbf{R} = \begin{bmatrix} 0 & 3/2 & 0 & 0 \\ 3 & 0 & 3/2 & 0 \\ 0 & 3 & 0 & 3/2 \\ 0 & 0 & 3 & 0 \end{bmatrix}$$

Both DTMCs and CTMCs can only model deterministic systems; they cannot model randomized distributed systems characterized by interleaving the concurrent processes. Markov Decision Processes (MDPs) are appropriate substitutes for Markov chains since they can be viewed as a variant of Markov chains with both probabilistic and nondeterministic choices.

Definition 2.3. *Over a set of atomic propositions AP, a model of MDP can be expressed as a tuple* $\{S,\ Act,\ \mathbf{P}\ ,\ I_{init},\ L,\ AP\}$, *where:*

- S, I_{init}, $L : S \to 2^{AP}$ *are as for DTMC*
- $\mathbf{P} : S \times Act \times S \to [0, 1]$ *is the transition probability function, such that for every state* $s \in S$ *and actions* $\alpha \in Act$, *we have* $\sum_{s' \in S} P(s,\ \alpha,\ s') \in \{0, 1\}$.
- *Act is a set of actions,* $\forall \alpha \in Act$, $\sum_{s' \in S} P(s,\ \alpha,\ s') = 1$

Figure 2.1 (c) is a graph example of a MDP model. $S = \{s_0, s_1, s_2, s_3\}$ is the set of states. $AP = \{r,\ f,\ s\}$ is the set of atomic proposition. $L(s_0) = \{\varnothing\}$, $L(s_1) = \{r\}$,

$L(s_2) = \{s\}$ and $L(s_3) = \{f\}$ are state labeling functions. The initial distribution I_{init} is the same as in our DTMC example. The transition probability functions $\mathbf{P}$ are: $\mathbf{P}(s_0) = \{(s_0, r1, s_1 \to 1)\}$, $\mathbf{P}(s_1) = \{(s_1, r1, s_2 \to 0.9), (s_1, r1, s_2 \to 0.1), (s_1, r2, s_1 \to 1)\}$, $\mathbf{P}(s_2) = \{(s_2, r2, s_2 \to 1)\}$, and $\mathbf{P}(s_3) = \{(s_3, r2, s_0 \to 1)\}$.

If there is only one element in *Act* set for all states in an MDP model, an MDP model becomes a Markov chain. Therefore, Markov chains are specific Markov decision processes.

Markov chains and Markov decision processes are widely used for constructing stochastic systems. The standard models of Markov decision processes with belief are Partially Observable Markov Decision Processes (POMDPs), which assume that agents cannot observe the current state. In POMDP, an agent maintains a probability distribution over possible observations for each action and resulting state and retains its goal to maximize expected future reward. To deal with the partially observable case, belief states [23] are introduced in a standard way as sets of states to model what the agent believes at different times. Updating the next belief state depends on the current belief state, the current action, and observations. POMDPs are widely studied in AI for their heuristic approaches to discovering optimal strategies that can be taken by agents, but they cannot observe independent knowledge relation, such as, agent i knows X and $X \Rightarrow Y$, so agent i knows Y. The formal definition is the following [62]:

Definition 2.4. *A model of POMDP is a tuple* $< S,\ Act,\ \mathbf{P},\ \Omega,\ \mathcal{O} >$*, where:*

- S*,* Act*,* $\mathbf{P}$ *are as for MDP*
- Ω *is finite set of observations that agents can explore of their world.*
- $\mathcal{O} : S \times Act \to \Pi(\Omega)$ *is the observation function. It presents a probability distribution* $\Pi(\Omega)$ *over possible observations for each action and resulting state.*

A Partially Observable Markov decision process (POMDP) is a special case of MDP in which the agent is in a partially observable environment. It has been used to model the uncertainty of knowledge and behavior for stochastic agents since the 1990s [23, 48, 61, 62]. POMDP have been widely adopted in machine learning [4, 89, 91], agent decision making [83], and robotic applications [63, 92].

2.2 Epistemic Models

An interpreted system is a framework based on a number of other possible states of affairs besides the true state of affairs. It was introduced by Fagin et al in [45]. The formalism of interpreted systems, which provides well-defined semantics to reason about time and knowledge in MAS, is frequently used in epistemic model checkers, such as MCMAS [76] and MCK[93].

Let $A = \{1, \ldots, n\}$ be a set of n agents in the system. Every agent $i \in A$ is associated with its local state set L_i, and possible actions set Act_i. Besides L_i and Act_i, there are also a set L_e and a set Act_e for the environment, a special agent for providing global variables and actions that all agents are able to access. Therefore, for the system, a set of global states $G \subseteq L_1 \times \ldots \times L_n \times L_e$ is the set of all possible tuples $(l_1, \ldots, l_n, l_e)$, and each tuple represents a computational state for the whole system. For each agent i, we also use a set of protocols $P_i : L_i \to 2^{Act_i}$ assigning a list of enabled actions to each local state. Associated with the environment is a protocol $P_e : L_e \to 2^{Act_e}$ that represents the functioning behavior of the environment agent. The probabilistic transition function T for the system can be defined as $T : G \times Act \times G \to [0, 1]$, where Act is the set of joint actions $Act \subseteq Act_1 \times \ldots \times Act_n \times Act_e$ that are performed by all the agents and environment respectively. Each agent is associated with a local probabilistic transition function $t_i \subseteq T$. Given a global initial state I, a set

of atomic propositions AP and an interpretation $V \subseteq G \times AP$, an interpreted system over probabilistic transition function is a tuple: $IS =< (L_i,\ Act_i,\ t_i,\ P_i)_{i \in A},\ I,\ V >$

The epistemic accessibility relation is defined to associate a Kripke model with a given interpreted system IS to express an epistemic model.

Definition 2.5. *The epistemic accessibility relations $\sim_i$ is a binary relation between two global states.* $(l_1,\ \ldots,\ l_n) \sim_i\ (l'_1,\ \ldots,\ l'_n)$ iff $l_i = l'_i$.

The epistemic accessibility relations are equivalence relations on W. Therefore, $\sim_i$ is reflexive, symmetric and transitive. Based on these relations, an epistemic Kripke model can be defined as follows:

Definition 2.6. *An epistemic Kripke model is a tuple* $M_{IS} = (W,\ I,\ R_t, \sim_1, \ldots, \sim_n, V)$, *where:*

- $W \subseteq G$: *the set reachable states of G,*
- $I \subseteq W$: *the set of initial states,*
- $R_t \subseteq W \times W$: *the temporal relation which is obtained using the protocols P_i and evolutions functions t_i*
- $\sim_i \subseteq W \times W$ *for* $i \in A$: *the epistemic accessibility relation for each agent* $i \in A$,
- $V \subseteq W \times AP$: *the evaluation relation as in IS.*

In the following section, we introduce how to evaluate Computation Tree Logic of Knowledge (CTLK) formulae for epistemic models.

2.3 Computation Tree Logic of Knowledge (CTLK)

CTLK was first put forward by Lomusico [72] and Penczek [85]. It uses CTL (proposed by Clarke and Emerson [32]) as basic temporal language and adds an epistemic component. The syntax of CTLK based on [85] is listed in Definition 2.7.

Definition 2.7. (Syntax of $CTLK$)

Let ϕ and ψ be CTLK formulae. We use $p, p_1, p_2, \ldots$ to range over the set of atomic propositions AP. Given a set of agents $A = \{1, \ldots, n\}$ as a group, the CTLK formulae follow BNF grammar:

$$\phi, \psi ::= true \mid p \mid \neg\phi \mid \phi \wedge \psi \mid AX\phi \mid AG\phi \mid A(\phi \mathrm{U}\ \psi) \mid K_i\phi \mid E_\Gamma\phi \mid C_\Gamma\phi \mid D_\Gamma\phi$$

The syntax above defines standard CTL operators and epistemic operators in CTLK. The notation of standard CTL operators in CTLK has the same meaning as in CTL. For example, the formula $AX\phi$ has the meaning of "in all possible paths in the next step ϕ holds," $AG\phi$ stands for "along all possible paths ϕ is always true.". $A(\phi \mathrm{U} \psi)$ represents that "in all possible paths at some point ψ holds and before then ϕ is true along the path." The epistemic operators include $K_i\phi$, $E_\Gamma\phi$, $C_\Gamma\phi$, and $D_\Gamma\phi$ and represent "agent i knows ϕ", "everyone in group Γ knows ϕ", "ϕ is common knowledge in group Γ", "ϕ is distributed knowledge in group Γ", respectively. Other basic modalities are defined in Table 2.1

Table 2.1: CTLK Basic Modalities Definition

Temporal Logic			Epistemic Logic		
$AF\phi$	$\equiv$	$A(\mathrm{true}\mathrm{U}\phi)$	$\overline{K}_i\phi$	$\equiv$	$\neg K_i \neg\phi$
$EG\phi$	$\equiv$	$\neg AF(\neg\phi)$	$\overline{D}_\Gamma\phi$	$\equiv$	$\neg D_\Gamma \neg\phi$
$EX\phi$	$\equiv$	$\neg AX\neg\phi$	$\overline{C}_\Gamma\phi$	$\equiv$	$\neg C_\Gamma \neg\phi$
$\phi \vee \psi$	$\equiv$	$\neg(\neg\phi \wedge \neg\psi)$	$\overline{E}_\Gamma\phi$	$\equiv$	$\neg E_\Gamma \neg\phi$

Let $M_{IS} = (W,\ I,\ R_t, \sim_1, \ldots, \sim_n, V)$ be an interpreted system model. Group Γ is a subset of M_{IS}. Epistemic relations based on the epistemic accessibility relations

and are defined as follows:

Definition 2.8. Epistemic Relations

The union of group $\Gamma's$ *accessibility relations is defined as* $\sim_\Gamma^E = \bigcup_{i\in\Gamma} \sim_i$. $\sim_\Gamma^C$ *is the transitive closure of* $\sim_\Gamma^E$. *The joint of* $\Gamma's$ *accessibility relations is* $\sim_\Gamma^D = \bigcap_{i\in\Gamma} \sim_i$.

Let s be a state, π be a path which is an infinite sequence of states related by transitions, i.e., $\pi = s_0, s_1, \ldots$. The $(i+1)th$ state in π is denoted $\pi(i)$, i.e., $\pi(i) = s_i$. $s \models \phi$ denotes "s satisfies ϕ" or ϕ is true in s. Let $\Pi(s)$ be a set of all the infinite paths starting at s in M_{IS}, ϕ and ψ be formulae of CTLK, the semantics of CTLK is shown in Definition 2.9.

Definition 2.9. (Semantics of CTLK)

$s \models p$ iff $V(s,p) = true$

$s \models \phi \wedge \psi$ iff $s \models \phi$ *and* $s \models \psi$

$s \models \neg\phi$ iff $s \not\models \phi$

$s \models AX\phi$ iff $\forall\pi \in \Pi(s)\ \ \pi(1) \models \phi$

$s \models AG\phi$ iff $\forall\pi \in \Pi(s)\ \ \forall i \geq 0\ \ \pi(i) \models \phi$

$s \models A(\phi \mathrm{U} \psi)$ iff $\forall\pi \in \Pi(s)\ \ \exists i \geq 0,\ \ \pi(i) \models \psi$ *and* $\forall 0 \leq j < i\ \ \pi(j) \models \phi$

$s \models K_i\alpha$ iff $\forall s' \in W$ *if* $s \sim_i s'$ *then* $s' \models \phi$

$s \models E_\Gamma\phi$ iff $\forall s' \in W$ *if* $s \sim_\Gamma^E s'$ *then* $s' \models \phi$

$s \models C_\Gamma\alpha$ iff $\forall s' \in W$ *if* $s \sim_\Gamma^C s'$ *then* $s' \models \phi$

$s \models D_\Gamma\alpha$ iff $\forall s' \in W$ *if* $s \sim_\Gamma^D s'$ *then* $s' \models \phi$

2.4 Probabilistic Temporal Logic

Hansson and Jonsson have put forward PCTL logic in [52]. PCTL language combines CTL [32] logic with probabilistic modalities [6, 52]. A PCTL formula is capable of

formulating conditions on the state of a Markov chain. Besides the standard propositional logic operators, PCTL also includes a probabilistic operator $Pr_{\nabla b}(\psi)$. In a formula including a probabilistic operator, such as $Pr_{\nabla b}(\psi)$, ψ is a path formula, whereas b is an interval of [0,1] and ∇b indicates a lower bound or upper bound on the probability (e.g $< b$, $\leq b$, $\geq b$, or $> b$). Unlike standard CTL, path quantifiers $\exists$ and $\forall$ are not valid in PCTL formulae; instead all path formulae are immediately preceded by the probabilistic operator $Pr_{\nabla b}$. We refer to Baier et al.'s definition in [6] as following in Definition 2.10:

Definition 2.10. (Syntax of PCTL)

Let ϕ be a state formula, ψ be a path formula, $\nabla \in \{<, \leq, \geq, >\}$, $b \in [0, 1]$, and $n \in \mathbb{N}$ be an integer. We use $p, p_1, p_2, \ldots$ to range over the set of atomic propositions Φ_p. The syntax of this language can be expressed as follows:

$$\phi ::= true \mid p \mid \phi \ \wedge \ \phi \ \mid \neg\phi \mid Pr_{\nabla b}(\psi)$$
$$\psi ::= \bigcirc\phi \mid \phi \ \mathrm{U}^{\leq n} \ \phi \mid \phi \ \mathrm{U} \ \phi$$

n indicates the maximum steps to achieve a specific state. There are no universal ($\forall$) and existential ($\exists$) path quantifiers in PCTL. Instead, the linear temporal operators $\bigcirc$ (next), U (until), and $\mathrm{U}^{\leq n}$ must follow the probabilistic operator $Pr_{\nabla b}$ immediately.

In order to express the fairness of concurrent probabilistic systems, Baier and Kwiatkowska proposed a probabilistic branching time logic PBTL in [7]. The difference between PBTL and PCTL is that PBTL allows universal ($\forall$) and existential ($\exists$) path quantifiers. Path quantifiers in PBTL formulae range over the adversaries and yield Markov chains. Therefore, all PCTL formulae can be expressed using PBTL formulae with a path $\exists$ quantifier. Referencing to [7], the syntax of PBTL can be defined as follows:

Definition 2.11. (Syntax of PBTL)[1]

Let ϕ be a state formula, ψ be a path formula, $\nabla \in \{<, \leq, \geq, >\}$, $b \in [0,1]$, and $n \in \mathbb{N}$ be an integer. We use $p, p_1, p_2, \ldots$ to range over the set of atomic propositions Φ_p. The syntax of this language can be expressed as follows:

$$\phi ::= true \mid p \mid \phi \ \wedge \ \phi \ \mid \neg\phi \mid Pr_{\nabla b}(\psi)$$
$$\psi ::= \exists \bigcirc \phi \mid \forall \bigcirc \phi \mid \phi \ \exists \mathrm{U}^{\leq n} \phi \mid \phi \ \forall \mathrm{U}^{\leq n} \phi \mid \phi \ \exists \mathrm{U} \phi \mid \phi \ \forall \mathrm{U} \phi$$

PBTL formulae can be evaluated over a PBTL structure, which is a tuple $(\mathscr{P}, I, AP, L)$ where $\mathscr{P} = (S, Steps)$ is a concurrent probabilistic systems and $L : S \rightarrow 2^{AP}$ is a labeling function which assigns to each state $s \in S$ a set of atomic propositions AP.

A path π is a nonempty (finite or infinite) sequence of states related by determinate transitions: $\pi = (s_0, s_1, \ldots)$. The $(i+1)th$ state in a path π is denoted $\pi(i)$. An adversary converts a nondeterministic system into deterministic one. The path quantifiers $\exists$ and $\forall$ involve quantification over adversaries. $Path^{\mathcal{A}}(s)$ is used to express the set of all path with the adversary $\mathcal{A}$ starting from state s. We use Adv to stand for set of adversaries for the system. $Prob\{\pi \in Path^{\mathcal{A}}(s) : \pi \models \psi\}$ is the probability of the all path such that $\pi \models \psi$ under the adversary $\mathcal{A}$. Let $a \in AP$ be an atomic proposition, ϕ be a PBTL state formula, and ψ be a PBTL path formula, $s \models_{Adv} \phi$ denotes "s satisfies ϕ under the adversaries set Adv" or "ϕ is true in s under the adversaries set Adv". $\pi \models_{Adv} \psi$ denotes "π satisfies ψ under the adversaries set Adv" or "ψ is true in π for the adversaries set Adv". The satisfiability of a PBTL formula at state $s \in S$ of a given concurrent system is defined inductively as follows.

[1] We modified the syntax of some operators to be compatible with PCTL.

Definition 2.12. (Satisfiability of PBTL)

$s \models_{Adv} true$ iff $\forall s \in S,\ s \models_{Adv} true$

$s \models_{Adv} p$ iff $p \in L(s)$

$s \models_{Adv} \phi_1 \wedge \phi_2$ iff $s \models_{Adv} \phi_1$ *and* $s \models_{Adv} \phi_2$

$s \models_{Adv} \neg\phi$ iff $s \not\models_{Adv} \phi$

$\pi \models_{Adv} Pr_{\triangledown b}(\exists \bigcirc \phi)$ iff $Prob\{\pi \in Path^{\mathcal{A}}(s) : \pi \models_{Adv} \bigcirc\phi\}\triangledown b$ for some $\mathcal{A} \in Adv$.

$\pi \models_{Adv} Pr_{\triangledown b}(\forall \bigcirc \phi)$ iff $Prob\{\pi \in Path^{\mathcal{A}}(s) : \pi \models_{Adv} \bigcirc\phi\}\triangledown b$ for all $\mathcal{A} \in Adv$.

$s \models_{Adv} Pr_{\triangledown b}(\phi_1 \exists \mathrm{U}^{\leq n} \phi_2)$ iff $Prob\{\pi \in Path^{\mathcal{A}}(s) : \pi \models_{Adv} \phi_1 \mathrm{U}^{\leq n}\phi_2\}\triangledown b$ for some $\mathcal{A} \in Adv$.

$s \models_{Adv} Pr_{\triangledown b}(\phi_1 \forall \mathrm{U}^{\leq n} \phi_2)\triangledown b$ iff $Prob\{\pi \in Path^{\mathcal{A}}(s) : \pi \models_{Adv} \phi_1 \mathrm{U}^{\leq n}\phi_2\}\triangledown b$ for all $\mathcal{A} \in Adv$.

$s \models_{Adv} Pr_{\triangledown b}(\phi_1 \exists \mathrm{U} \phi_2)\triangledown b$ iff $Prob\{\pi \in Path^{\mathcal{A}}(s) : \pi \models_{Adv} \phi_1 \mathrm{U}\phi_2\}\triangledown b$ for some $\mathcal{A} \in Adv$.

$s \models_{Adv} Pr_{\triangledown b}(\phi_1 \forall \mathrm{U} \phi_2)$ iff $Prob\{\pi \in Path^{\mathcal{A}}(s) : \pi \models_{Adv} \phi_1 \mathrm{U}\phi_2\}\triangledown b$ for all $\mathcal{A} \in Adv$.

$\pi \models_{Adv} \bigcirc\phi$ iff $\pi(1) \models_{Adv} \phi$

$\pi \models_{Adv} \phi_1\ \mathrm{U}^{\leq n}\phi_2$ iff $\exists 0 \leq k \leq n,\ \pi(k) \models_{Adv} \phi_2$ and $\forall 0 \leq i \leq k,\ \pi(i) \models_{Adv} \phi_1$

$\pi \models_{Adv} \phi_1 \mathrm{U}\phi_2$ iff $\exists k \geq 0,\ \ \pi \models_{Adv} \phi_1\ \mathrm{U}^{\leq k}\phi_2$

For the probabilistic operator, $s \models_{Adv} Pr_{\nabla b}(\psi)$ means that "over the adversary Adv from state s the probability that ψ is holds for an outgoing path satisfies is in the range of ∇b". The condition of adversaries can be either "there exists an adversary" or "for all adversaries" depending on which quantifier is used. For example, $s \models_{Adv} Pr_{<0.25}(true\exists \cup^{\leq 5} \phi)$ asserts that "there exists an adversary such that the probability that the system reaches ϕ being true within 5 steps of outgoing paths from state s is less than 0.25". From Definition 2.12, we can easily convert the PBTL formula $Pr_{\nabla b}\phi_1 \forall \mathrm{U} \phi_2$ to the identified PCTL formula $Pr_{\nabla b}\phi_1 \mathrm{U} \phi_2$; while $Pr_{>p}(\phi_1 \exists \mathrm{U} \phi_2)$ corresponds to $\neg Pr_{\leq p}(\phi_1 \mathrm{U} \phi_2)$. The reader can refer to [7] for more details.

2.5 Model Checking Algorithms in MAS

As mentioned earlier, model checking is a three-step process [32] (modeling, specification, and verification) to determine if a system satisfies a specification given as a temporal logic formula. Model languages are used to construct the system models. Logic languages are used to formalize the properties of the systems. Verification is the procedure of establishing if a given formula ϕ that is written in a particular logic and expresses a given properties of the systems is satisfied in a given model M. There are two input data: the formula ϕ and the model M. Two kinds of output result from model checking: either a satisfiable result, or a violated result with counterexamples. Some model checkers also provide simulations for locating the errors. Figure 2.2 from [6] shows the model checking process.

In the 1990s, Halpern and Vardi put forward in [51] an application of model checking within the context of the logic of knowledge. After that, several approaches have been proposed for model checking MAS. In [105], Wooldridge et al. proposed

an imperative programming language, MABLE to specify MAS along with a Belief-Desire-Intention (BDI) logic to express the properties. SPIN, an automata-based model checker, has been used to verify if the specified MABLE model satisfies the expressed properties. Another method based on the SPIN model checker has been developed by Bordini et al. [17] using AgentSpeak(F) Language, a BDI logic-based programming language [87]. SPIN generates C sources for a model checker in order to save memory and improve performance. There are many different approaches for executing model checking in MAS. Broadly speaking these different approaches can be classified into the categories listed below:

1. Automata-based approaches
2. Symbolic model checking algorithms
3. SAT-based approaches
4. Probabilistic model checking algorithms

In the following, we will discuss these model checking techniques.

2.5.1 Automata-Based Approaches

The automata-based model checking approaches were originally put forward by Vardi and Wolper in [96]. These approaches are used in checking Linear Temporal Logic formulae that can be represented by a nondeterministic Büchi automata (NBA). By finding a path π in the transition system TS with $\pi \models \neg\psi$, we can conclude that an error trace exists and $TS \not\models \psi$. Otherwise, $TS \models \psi$. An automaton is defined as Definition 2.13.

Definition 2.13. *A nondeterministic Büchi automata is a tuple* $\mathcal{A} = (Q, \Sigma, \delta, Q_0, F)$*, where*

- *Q is a finite states,*
- *Σ is finite alphabet, of initial states,*
- *$\delta \subseteq Q \times \Sigma \times Q$ is a transition relation,*
- *$Q_0 \subseteq Q$ is a set of initial states,*
- *$F \subseteq Q$ is a set of final (or accepting) states.*

The basic automata-based model checking algorithm is shown in Table 2.2

Table 2.2: Algorithm for Automata-Based Model Checking

Input:Transition system TS and formula ψ Output: "yes" or "no"
Construct an NBA $\mathcal{A}$ ϕ to represent the formula $(\neg\psi)$ Construct the product transition system $TS \otimes \mathcal{A}$ **if** there exists a path π in $TS \otimes \mathcal{A}$ satisfying the accepting condition of $mathcalA$ **then** return "no" and and an expressive prefix of π **else** return "yes" **fi**

Automata-based approaches can work with other techniques for model checking more complex systems. In [10], authors used an automata-based approach with tableau techniques to check communicating agent-based systems. Tableau techniques for model checking use *assertions* and tableau rules which are *proof rules* to verify whether the model M satisfies the formula ϕ. Assertions are typically of the form $s \vdash_M \phi$ and mean that state s in model M satisfies the formula ϕ. A set of tableau rules are used to prove the truth or falsity of assertions. Tableau-based algorithms work in a *top-down* or *goal-oriented* fashion, which is different from traditional proof systems that are usually bottom-up approaches. According to this approach, we start from a goal, and we apply a proof rule and determine the subgoals to be proven. Tableau rules are used in order to prove a certain formula by inferring when a state in

a Kripke structure satisfies such a formula. Therefore, a tableau-based algorithm saves on both time and space required for model checking because the algorithm searches only that part of the state space that needs to be explored to prove or disprove a certain formula. We now look closely at the tableau rules of the tableau-based algorithm in [10] for CTL^{*CA} propositional and universal formulae, which are just one part of CTL^{*CA} tableau rules. (CTL^{*CA} is an extended logic from CTL* for Communicative agents. More details of this logic and entire model checking algorithm can be found in [10].)

Table 2.3: Tableau Rules for CTL^{*CA} Propositional and Universal Formulae

Rule No.	Labels	Tableau Rules	Rule No.	Labels	Tableau Rules
$R1$	$\wedge$	$\frac{\psi_1 \wedge \psi_2}{\psi_1 \ \ \psi_2}$	$R2$	$\vee$	$\frac{\psi_1 \vee \psi_2}{\psi_1 \ \ \psi_2}$
$R3$	$\vee$	$\frac{E(\psi)}{\psi}$	$R4$	$\neg$	$\frac{\neg\psi}{\psi}$
$R5$	?	$\frac{?\psi}{\psi}$	$R6$	$\neg$	$\frac{A(\Phi)}{E(\neg\Phi)}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

Table 2.3 gives tableau rules of CTL^{*CA} for propositional and universal formulae. Tableau rules define a top-down proof system. Given a formula, we apply a tableau rule and determine the sub-formulae to be proven. Tableau rules are applied to a formula by proving all its sub-formulae. Labels of these rules are the labels of states in the automata constructed from a given formula. For Rule $R1$ in Figure 2.3 labeled by "$\wedge$" indicates that ψ_1 and ψ_2 are the two sub-formulae of $\psi_1 \wedge \psi_2$ so that we have to prove two children of the state satisfy ψ_1 and ψ_2 respectively. Rule $R3$ labeled by "$\vee$" indicates that ψ is the sub-formula to be proved in order to prove that a state satisfied $E(\psi)$. The syntactical operator"?" to express the tableau rule of the challenge action, which means that a given agent does not know whether the formula is true or not. According to Rule $R5$, the formula "$?\psi$" is satisfied in a state labeled by "?", if this state has a successor representing ψ.

2.5.2 Symbolic Model Checking

The traditional CTL model-checking procedure based on transition systems has an explicit enumerative representation per state. The number of states in transition systems grows exponentially with the number of components. This situation is called "the model checking state-explosion problem". Therefore, it is impossible to verify very large transition systems. There are several techniques to alleviate this problem, such as symbolic model checking based on Ordered Binary Decision Diagrams (OBDDs) and partial model checking algorithms. NuSMV [27], MCK [93], and MCMAS [76] are examples of model checkers using a symbolic model checking technique. NuSMV supports both Linear Temporal Logic (LTL) and computation tree logic (CTL). MCK works on a particular input model of synchronous interpreted systems of knowledge. The specification formulae in MCK can be either LTL or CTL augmented with knowledge. In MCMAS, similar to MCK, models are described in a modular language called Interpreted Systems Programming Language (ISPL). MCMAS supports a large set of specification languages, such as CTL, epistemic logics, and Alternating Time Logic (ATL). We usually use Ordered Binary Decision Diagrams (OBDDs) to symbolically express the states and transitions.

An OBDD model is a rooted, directed acyclic graph G that can be associated to a Boolean function $f(x_1, \ldots, x_n)$ by imposing a set of ordered, Boolean variables $x_1 < \ldots < x_n$ and by reducing the graph [18]. Figure 2.3 is an example of a reducing Boolean function $f = x_1 \vee (x_2 \wedge x_3)$. Since 1990s, OBDD techniques applied to model checking have been introduced in various papers [20, 80]. The model checking using OBDD associates the set of states and the transition relation to two OBDDs respectively. By comparing the two OBDDs it is possible to verify the formula. The details of this techniques are presented in [32].

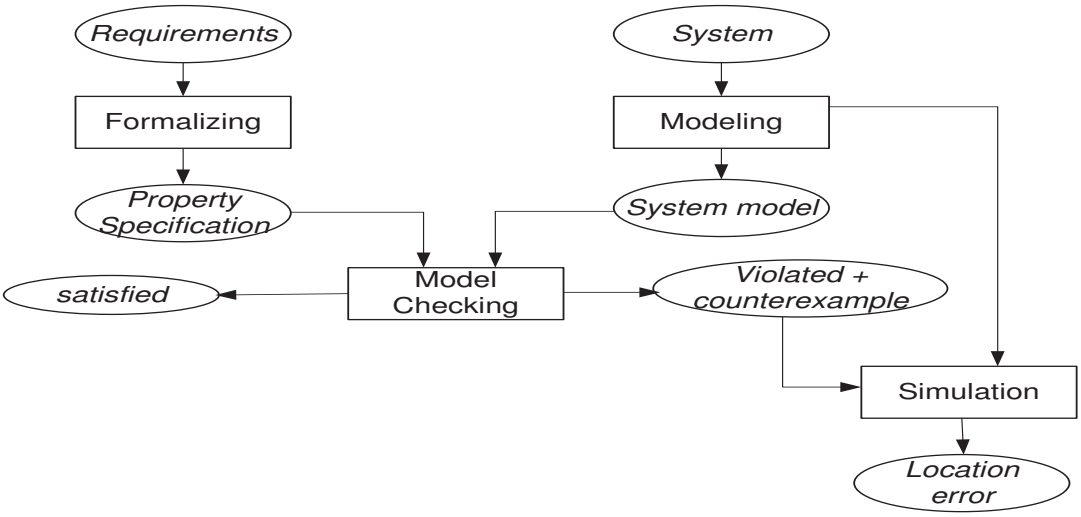

Figure 2.2: Schematic View of The Model-checking Approach from [6]

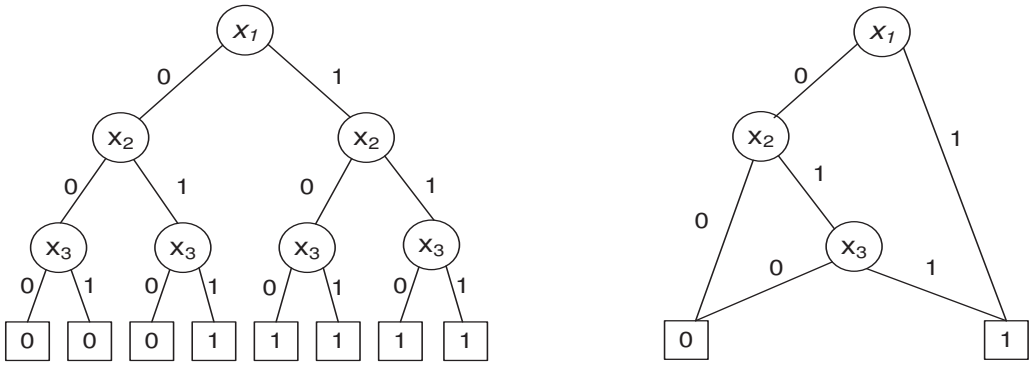

Figure 2.3: OBDD Example for $f = x_1 \vee (x_2 \wedge x_3)$

However, in probabilistic model checking, OBDDs are not sufficient to represent formulae and models because real-valued matrices and vectors are required. Thus, MTBDDs (multi-terminal binary decision diagrams) [30] are useful supplements. MTBDDs extend BDDs by allowing the representation of functions over Boolean vectors that can take any value instead of only 0 or 1. Like BDDs, an MTBDD is a directed acyclic graph. Figure 2.4 gives an example of an MTBDD over four Boolean variables x_1, x_2, x_3, and x_4. In fact, if we take Boolean variables $\{x_1, x_2\}$ to range over row indices and $\{x_3, x_4\}$ to range over column indices, the MTBDD in Figure 2.4 represents the transition probability function $\mathbf{P}$ of the associated DTMC is shown in Figure 2.1 (a)

2.5.3 SAT-Based Model Checking Algorithms

SAT-based model checking is a model checking technique that transform a model checking problem into a problem of satisfiability for propositional Boolean formulae (SAT). The SAT-based model checking algorithm first computes the satisfaction set $SAT(\phi)$ of all stats satisfying ϕ recursively based on parse tree of ϕ: the nodes of the formula ϕ parse tree represent the subformulae of ϕ. A state $s \models \phi$ if and only if $s \in SAT(\phi)$. Table 2.4 shows the basic idea of this algorithm for CTL formula ϕ [5].

There are bounded model checking and unbounded model checking. Bounded model checking is the first step in applying SAT procedures to symbolic model checking. It is based on the concept of bounded semantics. For Linear Temporal Logic (LTL) model M, given a formula ψ and a finite integer k, it can be proven that $M \models \psi$ if and only if there exists a finite integer k that formula ψ holds in M alone with a path of length k. In [15], Biere et al. proved that bounded model checking techniques can identify false formulae quicker than OBDD-based techniques if the bound value k is small. For CTL, Penczek et al. presented a bounded semantics for

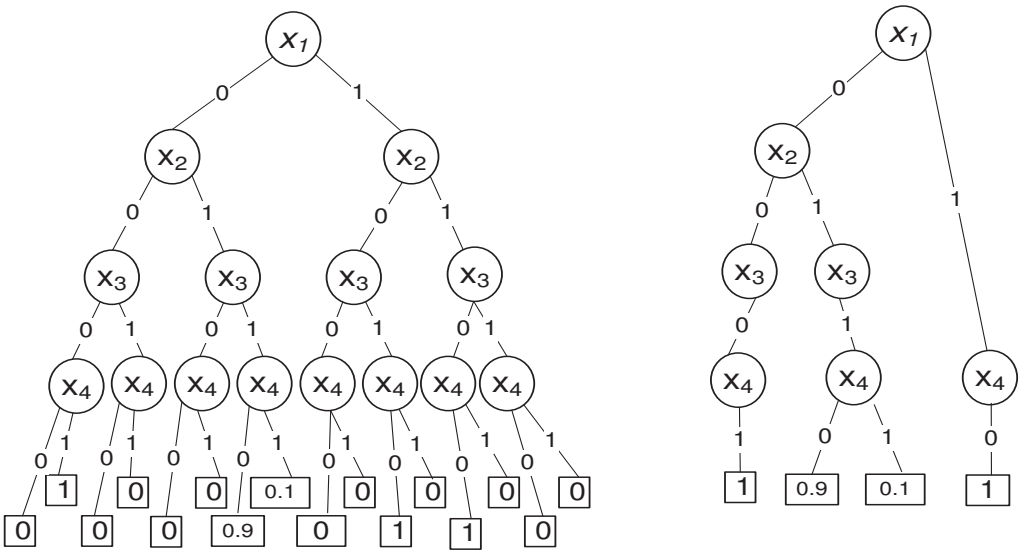

Figure 2.4: An MTBDD Example Figure 2.1 (a) DTMC model

Table 2.4: Algorithm for Basic Idea of CTL Model Checking

Input:Transition system TS and formula ϕ
Output: "yes" or "no"

******(Compute the sets $SAT(\phi) = \{s \in S | s \models \phi\}$) ******
for all $i \leq |\phi|$ **do**
for all $\psi \in Sub(\phi)$ with $|\psi| = i$ **do**
compute $SAT(\psi)$ from $SAT(\psi')$
******for maximal genuine $\psi' \in Sub(\psi)$ ******
od
od
if $s \in SAT(\phi)$
return "yes"
else
return "no"
fi

CTL by introducing ACTL, which restricts negation to atomic formulae only, and permits universally quantified temporal operators only. A detailed presentation of this approach can be found in [86].

A problem for the bounded model checking technique occurs when the bound value k is high or when formulae are true in a model. In these cases, the performance of an SAT-based technique decreases significantly. In order to solve this problem, McMillan present unbounded model checking in [81]. Unbounded model checking algorithms are similar to symbolic model checking: instead of representing Boolean formulae using OBDDs and comparing OBDDs, unbounded model checking algorithms translate model checking formulae into a satisfiable set for boolean formulae.

2.5.4 Probabilistic Model Checking Algorithms

Model checking techniques that we mentioned above only focus on the qualitative properties that guarantee systems absolutely, (i.e, that a certain bad event will never happen). In practice, however, many properties are not only important in terms of correctness but also of measurable performance; and systems are subject to unreliable and unpredictable behaviors as well. Therefore verifying quantitative properties are as important as verifying qualitative properties. Probability is widely used in the description and analysis of uncertain systems. Probabilistic model checking algorithms are designed for calculating the likelihood of the occurrence of certain events during the execution of the systems and checking both qualitative properties and quantitative properties.

Courcoubetis and Yannakakis have proposed a probabilistic model checking algorithm in [35, 36]. The models in probabilistic model checking algorithms are probabilistic. The simplest probabilistic models are *discrete-time Markov chains* (DTMCs),

which are introduced in 2.1. In DTMCs, models are specified the probability of making a transition from state s to a target state s', where the probabilities of reaching the target state from a given state must sum up to 1. PCTL introduced in 2.4 allows us to express both qualitative and quantitative properties. We now illustrate the probabilistic model checking algorithm for PCTL over DTMC.

The basic probabilistic model checking procedure is to compute the satisfaction set $SAT(\phi)$, which is done recursively using a bottom-up traversal of the parse tree of the formula ϕ. Computing the satisfaction set for the non-probabilistic operators is performed in exactly the same way as for propositional model checking algorithms. It is necessary to calculate the probability $Prop(s)$ that a path leaving each state s in order to compute SAT set for probabilistic operator. The probabilistic model checking algorithms are first presented in [35]. We use *bounded until* as an example for computing the probabilistic SAT set in the following section.

To compute $SAT(Pr_{\nabla b}[\phi_1 \cup^{\leq k} \phi_2])$, we first compute $SAT(\phi_1)$ and $SAT(\phi_2)$. Then we can identify the states that are for sure in the satisfaction set $S^{yes} = SAT(\phi_2)$, and for sure not in the satisfaction set $S^{no} = S \setminus (SAT(\phi_1) \cup Sat(\phi_2))$. We also need to classify the uncertain states, which may or may not be in the satisfaction set, $S^{?} = S \setminus (S^{yes} \cup S^{no})$. After we divide the states, we can compute the probability solution with a recursive equation:

$$Prob(s, \phi_1 \cup^{\leq k} \phi_2) = \begin{cases} 1, & \text{if } s \in S^{yes} \\ 0, & \text{if } s \in S^{no} \\ 0, & \text{if } s \in S^{?} \text{ and } k = 0 \\ \sum_{s' \in W} P(s, s') \cdot Prob(s, \phi_1 \cup^{\leq k} \phi_2), & \text{if } s \in S^{?} \text{ and } k > 0 \end{cases}$$

For detail algorithms of probabilistic model checking can be found in [52, 84].

2.6 Model Checking Tools

The techniques presented in Section 2.5 have been implemented in a number of model checkers. Many of them are distributed under the GNU Public License. This section briefly summaries five free model checkering tools and their programming and specification languages. The following reviewed tools are chosen because each one of them suits a different area that I focus on and I have used them during my research.

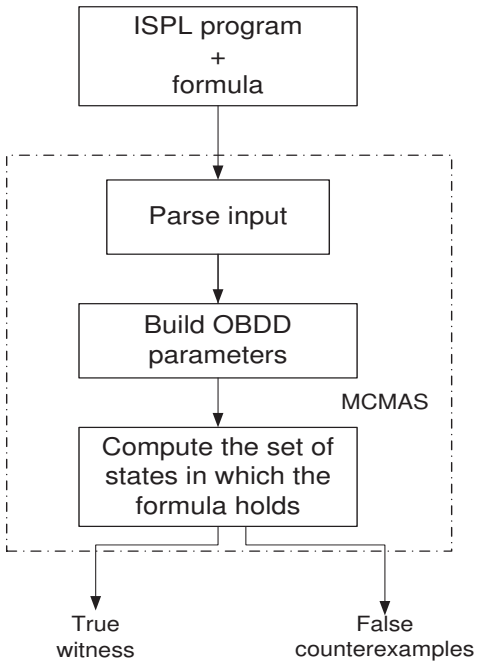

Figure 2.5: MCMAS Overview

MCMAS [76], a model checker for MAS, was developed in University College London. Its model checking algorithms are based on the Ordered Binary Decision Diagrams symbolic model checking technique. Figure 2.5 shows the basic procedure of MCMAS. The MCMAS program is based on a state-based formalism in which the behavior is defined by Kripke structures. This model checker can run on Linux, Max OS X, and Windows using Cygwin and be added as an Eclipse plug-in for user

graphical interface. MCMAS also is able to work interactive, step-by-step simulations. It can verify the properties of CTL and CTLK.

In MCMAS, MAS models are described by the Interpreted Systems Programming Language (ISPL), where the system can include two types of agents: an optional environment agent, which is used to describe boundary conditions and infrastructures, and standard agents. Each agent is composed of a set of local states, a set of actions, rules that describe which action can be performed by an agent, and evolution functions that describe how the local states of the agents evolve based on their current local state and agents' actions. ISPL can also be used to define atomic propositions, action formulae and the specification of properties to be checked. The highlight of MCMAS is that it is able to check epistemic properties, such as K_i, C_G, E_G, etc.

The first CTL model checker, EMC was announced by Clarke and Emerson in [28]. Based on EMC, Clarke et al. [29] constituted the SMV (Symbolic Model Verifier), which is an efficient CTL model checker based on a symbolic OBDD. **NuSMV** [27], developed jointly by ITC-IRST and Carnegie Mellon University via re-implementing and extending from SMV, is a software tool for the formal verification of finite state systems and the most widely cited model checker. It is based on symbolic model checking techniques for CTL and bounded model checking techniques for LTL.

NuSMV is a model checker in which the input language is designed to allow the description of finite state systems that range from completely synchronous to completely asynchronous. System models are translated into "MODULE". The system behavior is described in the module "*main*". A set of states and variables is defined in the "VAR" section, and how these states and variables change is constructed in "ASSIGN" section. Intended to describe finite state machines, the only data types in the language are finite ones, i.e. boolean, scalar and fixed arrays of basic data types. This is sufficient because NuSMV models basically are intended to describe

finite state machines. The verifying LTL and CTL specifications are also included in the module. Besides batch mode, NuSMV also provides a textual interaction mode so that users can activate various verifying steps. These steps can be invoked separately and can track back.

PRISM [66], developed in the Computing Laboratory at the University of Oxford, is a probabilistic symbolic model checker. It is a tool for formal modeling and analysis of systems which exhibit random or probabilistic behavior. Three types of probabilistic models, discrete-time Markov chains (DTMCs), continuous-time Markov chains (CTMCs) and Markov decision processes (MDPs) can be specified in the PRISM modeling language, which is a simple, state-based language. This language is based on the Reactive Modules formalism of Alur and Henzinger [3]. It offers limited extensions of these models with costs and rewards. The property specification language incorporates the temporal logics PCTL, CSL, LTL and PCTL*. Figure 2.6 shows the PRISM model checker procedure.

PRISM provides a Graph User Interface, which is implemented in Java. The core algorithms are mainly developed in C++. For state space representation, PRISM offers a choice among MTBDDs, "sparse matrices" and"hybrid" data structures. PRISM incorporates state-of-the art symbolic data structures and algorithms, based on BDDs (Binary Decision Diagrams) and MTBDDs (Multi-Terminal Binary Decision Diagrams). It also features discrete-event simulation functionality for generating approximate results for quantitative analysis.

MCK was developed at the School of Computer Science and Engineering at the University of New South Wales for model checking the logic of knowledge. MCK supports various ways of defining knowledge based on observations made by the agents: observation alone, observation and clock, synchronous and asynchronous perfect recall of all observations. Specifications can be described as either linear or branching

time temporal operators with epistemic and mu-calculus operators. The MCK model checker uses a concrete syntax designed to facilitate encoding of examples.

In MCK, MAS are modeled as a situation where agents interact in the context of an *environment*. Every agent has its local states and is capable of performing certain actions in the environment. Agents performs actions based on their protocol that describes the allowable choices of the next action at each point of time. System states include all environment states and agents' local states. Properties are expressed with specification formulae.

CWBNC is an enhanced version of the Concurrency Workbench of the New Century (CWB-NC) [33, 106]. It was developed by the Concurrency Workbench project group. This model checker supports two different temporal logics, the modal mu-calculus and GCTL*. It allows modeling concurrent systems using the process algebra Calculus of Communicating Systems (.ccs), Synchronous Calculus of Communicating Systems (.sccs), and CCS with prioritized actions (.pccs), with timed actions (.tccs). xCCS language is a paradigmatic process algebras language, which is a prototype specification language for reactive systems. For this reason, xCCS language can be used not only to describe implementations of processes, but also specifications of their expected behaviors.

In CWBNC, the specifications and formulas are first "compiled" into automata, then transformed into a type of deterministic automata. Therefore, this model checker only can check a finite state machine containing less than 60,000 reachable states.

In Table 2.6, we summarize these four model checkers.

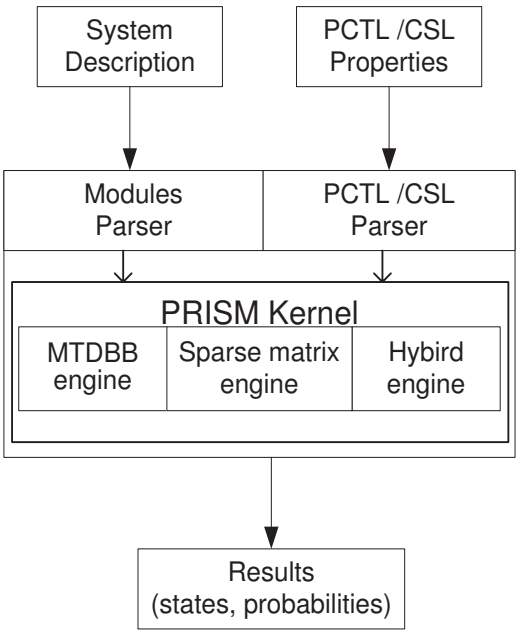

Figure 2.6: PRISM Overview

Table 2.5: Comparison of Model Checkers

Model checker	Model checking techniques	System model language	Properties language	platform
$MCMAS$	$OBDD$	$ISPL$	CTL $CTLK$	Window (Cygwin), Linux, Mac OS.
$NuSMV$	$BDD-based$ $SAT-based$	$.smv$	LTL CTL	Window
$PRISM$	BDD $MTBDD$	$DTMC$ $CTMD$ MDP	$PCTL$ $PCTL^*$ $CSL,$ LTL	Window (Cygwin), Linux, Mac OS.
MCK	OBDD bounded model checking	$.mck$	a combination of CTL* with epistemic and mu-calculus operators	Linux
$CWBNC$	Automaton-based	CCS $SCCS$ $PCCS$ $TCCS$	$mu-calculus$ $GCTL*$ CTL	Windows, Linux

Chapter 3

Related Work

Our work mainly involves three areas of research: 1) a framework for representing probabilistic-epistemic systems; 2) logics for expressing probabilistic-epistemic properties; and 3) approaches for model checking probabilistic-epistemic properties. In the rest of this section, we will compare our work with other researchers' work in these relevant areas.

3.1 Models

One main framework for representing epistemic systems is with interpreted systems formalism [45]. Reasoning about knowledge and time with interpreted system formalism is thoroughly investigated and extensively used in specifying and verifying MAS [34, 75, 85, 95]. With interpreted systems formalism, we can easily develop various epistemic modalities based on different agents' accessibility relations. For illustration, with agent epistemic accessibility relations, a knowledge modality is introduced to represent an individual agent's knowledge; while with a group of agents' epistemic accessibility relations, we can define distributed knowledge in the group. Recently, some researchers have enriched interpreted systems with probability in order to specify

uncertain agent systems [50, 56, 57, 102].

Most work on interpreted system frameworks only focus on epistemic systems [9, 34, 77, 75, 85, 95], while uncertainty of knowledge is not considered. Belardineli and Lomuscio first investigated extending interpreted systems into the first order logic in [8]. They defined quantified interpreted systems (QIS) to enable the use of quantifiers on epistemic models by endowing each structure with a domain of individuals within a first-order temporal epistemic logic. However, in their proposal, the stochastic behavior of agents and probabilistic properties are not considered. Also, the first order logic is not very expressive; for example, there is no first-order sentence that defines the finite structures, which are widely used in the model checking area. Another drawback for the QIS is that the decidability has not been studied yet.

Researchers in [56, 57] have integrated interpreted systems with partial observation techniques to express stochastic epistemic systems. The Partially Observable Markov Decision Processes (POMDPs) [23] provides another framework for reasoning about knowledge and are mainly used to represent uncertain knowledge of agents. POMDPs are a generalization of Markov Decision Processes (MDPs) and have been used to model the incomplete/uncertain knowledge and behavior for stochastic agents since the 1990s [23, 48, 61]. In the past decade, POMDPs have been extended into self-learning and reaction areas, such as machine learning [4, 89], robotic applications [63, 92], etc. In the POMDP-based framework, agents only partially observe the underlying states and maintain a probability distribution over the set of belief states. Belief states are computed based on a set of observations.

Both Huang and his colleagues [56, 57] and our work are based on interpreted systems. Unlike our work that integrates interpreted systems into Markov chains, Huang el al. proposed interpreted partially observed discrete-time Markov chains (PO-DTMCs) in [56]. PO-DTMCs are generalized from POMDPs with deterministic

choice of actions. This approach is based on partial observations with assumption of synchronous with perfect recall. In their method, the environment is a special agent that includes the set of states. Other agents observe the environment and perform actions based on their observations. Every agent, including the environment agent, has its own probability transition function for each accessible state based on its accessibility relations (the environment agent's transition function will be the system probability transition function). Probabilistic knowledge is expressed by a rational linear combination of every agents' probabilities in the system. The main problem is that the probabilities associated with accessible states are not part of the system, but part of the agents' programs and it is not clear how an agent obtains those probabilities. Practically, an agent can see which state is equivalent to or indistinguishable from the current state, but it is hard to see by how much the state is indistinguishable. Nevertheless, probabilistic knowledge in our approach is captured in a simpler and more practical way as a direct and inherent extension of non-probabilistic knowledge where the probability only depends on the number of equivalent states, as all the states are equally accessible.

These two frameworks both have advantages and disadvantages. Interpreted systems are widely used in modeling epistemic systems without uncertainty because interpreted systems provide a natural and elegant way of capturing the philosophical foundations of knowledge using a possible and accessible world. The specification of uncertain knowledge of MAS with interpreted systems is, however, still in an early stage of research. On the other hand, POMDPs, based on Markov Decision Processes, manipulate uncertainty intrinsically. POMDPs have been widely studied in artificial intelligence to discover optimal strategies that can be taken by an agent. Nevertheless, a POMDP models the relationship between an agent and its environment. POMDPs have been used in single agent systems mainly to find the best solution or strategy for

the agent. The algorithm for solving POMDP runs in exponential time in the size of the model's actions and observations. Furthermore, the dimensionality of the belief space grows with the number of states. In [102], we previously compared our work based on interpreted systems modeling with Huang et al.'s work [56] based on PO-DTMC modeling. The results show that our method performs better than Huang's in verifying the example protocol (see Chapter 4).

Besides those two main frameworks for representing and reasoning about epistemic systems, Delgado and Benevides in [39] specify each individual agent in MAS by a homogeneous DTMC with synchronization actions. In their special DTMC model, a state either has a synchronized action with probability 1 to represent agent behavior or regular probabilistic transitions. Agents collaborate with these synchronization actions. Then they formalize the composition of two DTMCs, a DTMC with regular probabilistic transitions and a DTMC with agents' synchronized actions, in a MDP to represent the behavior of the overall MAS. The drawback of this method is that it limits the agents' behavior by unifying all agents actions into one DTMC structure. In addition, only using global states for overall MAS constraints influences every agent's behavior.

3.2 Logics

Logics of probability and knowledge are highly related to our research. Both epistemic logic and probabilistic logic are multimodal logics that combine two types of modalities. Computation Tree logic of Knowledge (CTLK), proposed by Penczek and Lomuscio in [85], extends from CTL (introduced by Emerosn and Clarke in [44]) by adding epistemic modalities and is used in many MAS to specify epistemic properties. CTLK formulae are associated with interpreted systems. The basic temporal

modalities are defined as usual, while epistemic modalities are based on epistemic accessibility relations similar to our definitions. They also developed an OBDD-based symbolic model checker MCMAS [78] that can verify agents' knowledge in MAS. Meyden et al. in [95] define a language for reasoning knowledge and time in systems with perfect recall. Their language is applicable to interpreted systems as well. Unlike CTLK that extends the branching time logic CTL, their language is based on linear time. MCK [47] is designed for verifying epistemic properties of MAS and supports several different ways of defining agents' knowledge, such as using observation alone or observation and clock with either synchronous or asynchronous perfect recall of all the observations. Both CTLK and language presented by Meyden et al., however, do not consider probabilistic behaviors. Therefore, these languages cannot express agents' uncertain knowledge.

On the other hand, Probabilistic logics include probabilistic modalities and temporal modalities. This kind of logic has been used with Markov chains (determinism) or Markov Decision Processes (nondeterminism). Hansson and Jonsson in [52] have put forward probabilistic computation tree logic (PCTL) which focuses only on non-epistemic probabilistic properties. Path quantifiers $\exists$ and $\forall$ are not valid in PCTL formulae. However, Baier and Kwiatkowska proposed a probabilistic branching time logic PBTL which allows path quantifiers to verify probability with fairness constraints in [7]. The quantifiers $\exists$ and $\forall$ in PBTL range over the adversaries; thus, these quantifiers are only involved in deterministic choices. Another probabilistic logic example comes from Jamroga in [59], Markov temporal logic MTL_x, an extension of the "Discounted CTL" (DCTL), which uses a discount factor to achieve the probabilistic factor. He introduced MTL_0 for Markov chains and MTL_1 for Markov Decision Processes. $\text{MTL}_x(x \in \{0, 1\})$ allows agents to perform flexible reasoning about their

outcomes in stochastic environments. Unlike other probabilistic logic using probabilistic transition functions to present probabilistic choices, Jamorga uses *utility fluent* to present the truth values for Markov chains and MDPs. The probabilistic logics mentioned in this paragraph only focus on quantitative or probabilistic properties not on the epistemic ones. Consequently, neither epistemic logics nor probabilistic logics are suitable for uncertain knowledge of MAS.

Dealing with uncertainty or incompleteness within distributed knowledge bases has been recently addressed by some researchers in [22, 39, 56, 57, 107]. Logics that they proposed are more or less similar to our logic, but our logic is more expressive and better suited to epistemic-probabilistic systems. Huang el al. put forward Alternating-Time Temporal Logic (PATL*) in [56] to account for incomplete information in multi-players synchronous games. The syntax of this logic is as follows:

$$\phi ::= p|\ \neg\phi \mid \phi_1 \wedge \phi_2 \mid X\phi \mid \phi_1 \mathrm{U} \phi_2 \mid \langle\langle A\rangle\rangle^{\bowtie d}\phi$$

where p is an atomic proposition, and $A \subseteq Agt = \{1, \ldots, n\}$ is a player, d is a rational constant in $[0, 1]$, and $\bowtie$ is a relation symbol in the set $\{\leq, <, \geq, >\}$. The only difference from regular temporal logic formula is $\langle\langle A\rangle\rangle^{\bowtie d}\phi$, which expresses that players in A can collaborate to enforce the fact ϕ with a probability in relation $\bowtie$ with constant d. The semantics of this logic is associated with probabilistic interpreted systems and uses partially observed probabilistic concurrent game structure where players have perfect recall memory over observations. The logic offers the ability to reason about strategies on incomplete information over games involving multiple players. The paper proves that the model checking problem of PATL* is in general undecidable, which precludes the use of this logic to verify scalable concrete applications.

Delgado and Benevides[39] modeled MAS using DTMC with synchronization actions and defined K-PCTL logic to specify the properties. K-PCTL, an epistemic extension of the probabilistic CTL temporal logic, allows epistemic and temporal

properties as well as likelihoods of events. The syntax of K-PCTL is as follows:

$$\phi ::= true |\ a \mid \neg\phi \mid \phi \wedge \phi \mid \mathcal{P}_{rel\ p}[\psi] \mid K_i\phi \mid C_G\phi \mid E_G\phi$$
$$\psi ::= \phi \mid \phi \mathrm{U}^{\leq k}\phi \mid \phi \mathrm{U}\phi$$

where a is an atomic proposition, $rel \in \{\leq, <, \geq, >\}$, $p \in [0, 1]$, i is an agent in the system, and $k \in \mathbb{N}$. As shown in the syntax, K-PCTL can only express probabilities over path formulae; therefore, K-PCTL is not able to specify the uncertainty of the knowledge. Our PCTLK logic overcomes this limitation by allowing probabilities over knowledge operators (for simplifications, we omit group knowledge operators which include in the previous paper [102]). Thus, our logic is more expressive than K-PCTL because our PCTLK logic not only expresses probabilities on path formulae, but also includes probabilities of knowledge. The semantics of this logic is associated with MDP models augmented with the accessibility relations so that probabilities over paths can be defined and classic knowledge formula can be captured. However, probabilities of knowing cannot be captured because accessibility transitions are not probabilistic. Our approach surmounted this drawback by integrating probabilistic interpreted systems into Markov chains. Accessibility transitions are quantified during the integration so that they are probabilistic.

In contrast to K-PCTL, which focuses only on probability in path formulae, Cao in [22] proposed a probabilistic epistemic temporal logic, called PETL, which only includes probability of knowledge modalities but not of temporal modalities. Besides regular temporal formulae, he introduced probabilistic epistemic formulae to illustrate, $K_a^p\phi$, $E_\Gamma^p\phi$, and $C_\Gamma^p\phi$ (see [22] for syntax definition details). $K_a^p\phi$ means that agent a knows the probability of ϕ is greater than or equal to p. $E_\Gamma^p\phi$ and $C_\Gamma^p\phi$ are group epistemic formulae which express that every agent in Γ knows the probability of ϕ is greater than or equal to p and "the probability of ϕ is greater than or equal to p" is common knowledge by every agent in Γ. The main drawback for PETL is

that it cannot express properties like "in the future ϕ holds in p probability" and "the probability that an agent knows ϕ is p". In our PCTLK logic, probabilities can be expressed over the whole formula, which allows us to state these properties. That is, PCTLK can express both probability of paths (probability over the next, until, and global operators) and the probability of knowing, which is on top of properties about knowing the probability of formulae that PETL can express. Furthermore, PETL is based on LTL, while our logic is an extension of CTL. LTL and CTL are incomparable in terms of expressiveness, which means one cannot be a subset of the other. The semantics of PETL is associated with a probabilistic epistemic temporal model that includes the set of the global states, a total binary (successor) relation on global states, epistemic accessibility relations, a probability function for every agent, and a valuation function that assigns to each state a set of propositional variables that are assumed to be true at the state. The semantics of probabilistic knowledge is then defined using the probability function that is associated to each accessibility transition. The main difficulty with this definition is the computation of those probabilities over accessibility transitions, which are not part of the system being checked, but rather part of the agent's accessibility relations. It is not clear how to define or compute these probabilities. In our semantics, however, we assume that all the accessible states from a given state are equally accessible. Thus, accessibility transitions are not probabilistic. Nevertheless, probabilistic knowledge is naturally defined by computing the number of accessible states that satisfy the knowledge over the total number of accessible states.

Zhao et al. extended this logic to consider probabilistic aspects of systems and introduced the probabilistic temporal logic of knowledge called PTLK in [107]. PTLK can expresses probability on both path formulae and epistemic formulae. To express probability of path, PTLK modified PETL formulae "next" ($\bigcirc\phi$) and "until" ($\phi\mathbf{U}\phi$)

as $X^{\triangleright p}\phi$ and $\phi \mathbf{U}^{\triangleright p}\phi$ respectively, where $\triangleright \in \{>, \geq, <, \leq, =\}$, $p \in [0, 1]$ is a real number. However, PTLK still cannot specify "the probability that an agent knows ϕ is p". Also the probabilistic Kripke structure (PK), which is associated with PTLK semantics, is very similar to a probabilistic epistemic temporal model: they use the probability function associated with each accessibility transition to define probabilistic knowledge, as in PETL. Therefore, PTLK still has the same drawback as PETL.

There is a new trend that uses combinations of logics [64] to express many aspects of MAS, such as knowledge and time, knowledge and probability, real-time and knowledge, etc., to avoid repeating the implementation of many different verification systems. The component logics refer to key aspects of MAS, including classic temporal logics (CTL, LTL, etc.), belief/knowledge logics (model logics KD45, S5, etc.), logics of goals (modal logics KD, etc.), probabilistic temporal logics (PCTL, etc.), and real-time temporal logics (TCTL, etc.). The component logics can be combined in three ways:

1. Temporalisation, which adds a temporal dimension to another logic system;

2. Fusion (independent join), which is obtained by the union of the respective sets of connectives and the union of the formation rules of both logics [46]; and

3. Product (join), which produces higher-dimensional temporal logics by combining lower-dimensional temporal logics.

With these three combination methods, the syntax of combinations of logics is possible to cover all aspects of MAS. In their paper, Konur et al. provide a generic model checking algorithm, which synthesizes a combined model checker from the model checkers of simpler component logics. The authors do not clearly explain, however, how the semantics of those combined logics are associated with models. This issue affects the future implementation. Therefore, the realistic application of combinations

of logics still needs to be demonstrated. In short, combinations of logics still needs to be explored further.

3.3 Model Checking Approaches

Many model checking techniques have been developed to overcome the state explosion problem of model checking, such as symbolic techniques [19], reduction techniques [31], bounded model checking [14], etc. In order to use the existing probability model checker PRISM, we exploit reduction techniques to reduce model checking PCTLK to model checking PCTL. The foundation for reduction techniques is that a formula is true in the abstract system, if and only if (iff) it is true in the original system [31]. In addition to our method that transforms models and formulae into widely used models and formulae, there are also other reduction techniques, such as probabilistic abstraction techniques [107], approximate probabilistic techniques [54], and symmetry reduction techniques [34].

Hérault et al. in [54] presented a randomized algorithm that allows the efficient approximation of the satisfaction probability of monotone properties on probabilistic systems. They developed an approximate probabilistic model checker APMC to implement their method in [54]. The essentially positive fragment (EPF) of LTL was defined to express only monotone properties, which means a formula ϕ holds in a path σ if and only if the formula ϕ holds in any path σ^+ of which σ is a prefix. Then they denote $Prob[\psi]$ as the measure of the set of paths in the probabilistic transition system. They generate random paths in the probabilistic space underlying the DTMC structure of depth k and compute a random variable which estimates $Prob[\psi]$. They have used this approach to verify extremely large systems such as the Pnueli and Zuch's 500 dining philosophers. This approach however, can be used only on EPF of

LTL and does not work with epistemic properties.

Zhou et al. introduced the logic PTLK (probabilistic temporal logic of knowledge) and proposed an abstraction procedure for model checking PTLK in [107]. Their abstraction approach is based on partitioning the state space into several equivalence classes which consist of the set of abstract states. After partition, the probability distribution between these partitions is an interval. Every formula will correspond to a probability function over the interval. However, no experiments were implemented to show the reduction rate of their approach.

Cohen et al. used a symmetry reduction technique to verify temporal-epistemic logic CTLK in [34]. Only agent symmetries were considered. By permuting agent names along the epistemic accessibility relation, they reduced the interpreted systems semantics into a counterpart semantics. The approach exploited agent symmetries to reduce the initial states. Therefore, after reduction, there is a single representative for a group of symmetric initial states. The experimental results with the muddy children show significant reductions in verification time and states. However, uncertainty has not been considered. In future work, we intend to explore the extension of this method to the probabilistic model checking field.

There are model checking tools designed for the verification of epistemic and temporal properties of MAS, for example MCK [47], and MCMAS [78]. MCK supports agents' knowledge in both observation alone and observation with clock for synchronous or asynchronous perfect recall of all the observations. It can be used to verify either linear or branching time temporal logics. while MCMAS only supports branching temporal logic and CTLK. There are also some model checkers developed for verifting probabilistic specifications, like PRISM [69] and ProbVerus [53]. PRISM is a probabilistic symbolic model checker that can verify PCTL, CSL, LTL and PCTL*

formulae, as well as extensions for quantitative specifications and costs/rewards formulae. ProbVerus is an extension of Verus [21], which combines symbolic model checking techniques and quantitative algorithms for computing minimum and maximum time delay between two events. ProbVerus can be used to verify PCTL formulae on fully probabilistic systems. However, there are no model checking tools specifically for epistemic-probabilistic properties. We used the reduction technique to convert probabilistic-epistemic properties into quantitative properties so that we are able to use the PRISM model checker to check these epistemic-probabilistic properties.

Chapter 4

Probabilistic Computation Tree Logic of Knowledge

This chapter is mainly from the paper published in *Knowledge-based systems* [102]. In Section 4.1, which answers our first research question, we present the models and introduce probabilistic interpreted systems. We define a new logic PCTLK in Section 4.2 to answer our research question 2 and state syntax and semantics of PCTLK. Section 4.3 and Section 4.4 provide answers for research questions 3 and 4. In Section 4.3, we explain how model checking PCTLK can be reduced to model checking PCTL. We implement our approach with PRISM [66] and apply it to a case study in Section 4.4.

4.1 Probabilistic Interpreted Systems

Let $A = \{1, \ldots, n\}$ be a set of n agents in the system. Every agent $i \in A$ is associated with its local state set L_i, and possible actions set Act_i. A set of global states $\Gamma \subseteq L_1 \times \ldots \times L_n$ is the set of all possible tuples $(l_1, \ldots, l_n)$, and each tuple represents a computational state for the whole system. If we assume that all actions have even

chance, we can map actions to the probabilistic transition function T for the system. T is defined as $T : \Gamma \times Act \times \Gamma \rightarrow [0,1]$, where $Act \subseteq Act_1 \times \ldots \times Act_n$ is the set of actions that are executed by agents in the system for collaboration, such that for every global state $\gamma \in \Gamma$, $\sum_{\gamma' \in \Gamma} T(\gamma, \alpha^{\gamma\gamma'}, \gamma') = 1$, where $\alpha^{\gamma\gamma'} \in Act$ is the action labeling the transition from γ to γ'. Each agent is associated with a local probabilistic transition function $T_i : L_i \times Act_i \times L_i \rightarrow [0,1]$, such that for every local state $l_i \in L_i$, $\sum_{l'_i \in L_i} T_i(l_i, \alpha^{l_i l'_i}, l'_i) = 1$ for $i \in A$, where $\alpha^{l_i l'_i} \in Act_i$ is the agent i's action labeling the transition from l_i to l'_i. For $l_i \in \gamma$, $l'_i \in \gamma'$, the probabilistic transition function T for the system can be calculated by Equation 4.1 as follows:

$$T(\gamma, \alpha^{\gamma\gamma'}, \gamma') = \eta \prod_{\substack{i \in A \wedge \\ l_i \in \gamma \wedge \\ l'_i \in \gamma'}} T_i(l_i, \alpha^{l_i l'_i}, l'_i) \tag{4.1}$$

where η is a normalizing factor that forces transitions fit for probability distribution $\sum_{\gamma' \in \Gamma} t(\gamma, \alpha^{\gamma\gamma'}, \gamma') = 1$ for every global state γ. A global initial distribution I_{init} expresses how the system starts and satisfies $\sum_{\gamma \in \Gamma} I_{init}(\gamma) = 1$. Definition 4.1 defines the formal models of PCTLK M_{PIS}.

Definition 4.1. Models M_{PIS} of PCTLK

Over a set of atomic propositions AP, a model M_{PIS} is a tuple: $M_{PIS} = (W, \mathbf{P_t}, I_{init}, \sim_1, \ldots, \sim_n, V)$ where:

- *$W \subseteq \Gamma$ is the set of reachable states. A state w is reachable if and only if there exists a sequence of transitions from an initial state to w such that all of the transitions have probability greater than 0.*
- *$I_{init} : W \rightarrow [0,1]$ is the initial distribution of the model, such that: $\sum_{w \in W} I_{init}(w) = 1$.*
- *$\mathbf{P_t} : W \times W \rightarrow [0,1]$ is the transition probability function defined by $\mathbf{P_t}(w, w') =$*

p ($p \in [0,1]$) if and only if there exists a collaboration action $(a_1, \ldots, a_n) \in Act$ such that $\sum_{i \in A} T_i(w, a_i, w') > 0$ and the value of $\mathbf{P_t}$ is equal to $t(w, \alpha^{ww'}, w)$ in probabilistic interpreted systems. For all $w \in W$ we have: $\sum_{w' \in W} \mathbf{P_t}(w, w') = 1$.

- *$\sim_i \subseteq W \times W$ is the epistemic accessibility relation for the agent i, such that for two global states $(l_1, \ldots, l_n)$ and $(l'_1, \ldots, l'_n)$, we have: $(l_1, \ldots, l_n) \sim_i (l'_1, \ldots, l'_n)$ iff $l_i = l'_i$.*

- *V is a global state labeling function $V : W \to 2^{AP}$.*

The initial distribution I_{init} can be viewed as a column vector of $|S|$ rows where $|S|$ is the cardinality of S $(I_{init}(s))_{s \in S}$, in which the value of every row represents the probability that the corresponding state is an initial state.

The transition probability function $\mathbf{P_t} : W \times W \to [0,1]$ can be represented by the matrix $(\mathbf{P_t}(s,t))_{s,t \in W}$. The probabilities of moving from state s to its successors are shown on the rows $\mathbf{P_t}(s,.)$ of the matrix, while the probabilities of entering state s from other states are shown on the columns $\mathbf{P_t}(.,s)$ of the matrix.

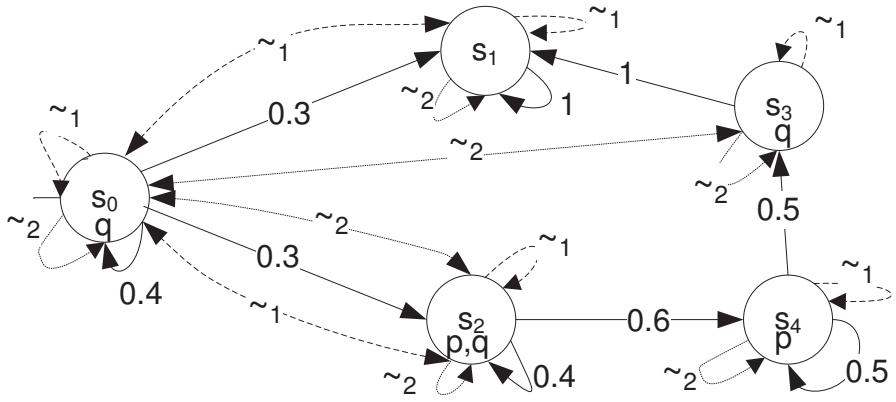

Figure 4.1: Model M_{PIS}

Let us consider the example illustrated in Fig 4.1 showing an M_{IPS} model where two agents are included. $W = \{s_0, s_1, s_2, s_3, s_4\}$ is the set of reachable states.

$AP = \{p, q\}$. The labeling function V is: $V(s_0) = \{q\}$, $V(s_1) = \{\varnothing\}$, $V(s_2) = \{p, q\}$, $V(s_3) = \{q\}$, and $V(s_4) = \{p\}$. For epistemic accessibility relations, we have: $\{(s_0, s_0), (s_0, s_1), (s_0, s_2), (s_1, s_0), (s_2, s_0), (s_1, s_1), (s_2, s_2), (s_3, s_3), (s_4, s_4)\} \subseteq \sim_1$ and $\{(s_0, s_0), (s_0, s_2), (s_0, s_3), (s_2, s_0), (s_3, s_0), (s_1, s_1), (s_2, s_2), (s_3, s_3), (s_4, s_4)\} \subseteq \sim_2$

The initial distribution I_{init} and the transition probability function $\mathbf{P_t}$ viewed as a 5×5 matrix are as follows:

$$I_{init} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \qquad \mathbf{P_t} = \begin{bmatrix} 0.4 & 0.3 & 0.3 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0.4 & 0 & 0.6 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.5 & 0.5 \end{bmatrix}$$

4.2 Epistemic-Probabilistic Logic

Specifications for interpreted DTMC models M_{PIS} can be expressed in PCTLK (Probabilistic Computation Tree Logic of Knowledge), which combines CTL logic [32], epistemic logic [45], and probabilistic logic [6, 52]. PCTLK can be used to reason about probabilistic knowledge and specify properties of probabilistic-epistemic MAS.

A PCTLK formula is capable of formulating conditions on a state of an epistemic Markov chain. Besides the standard propositional logic operators, PCTLK also includes the probabilistic operator Pr. Unlike standard CTL, path quantifiers $\exists$ and $\forall$ are not valid in PCTLK formulae, all path formulae are immediately preceded by the probabilistic operator Pr.

4.2.1 Syntax of PCTLK

PCTLK is comprised of three types of formulae: state formulae ϕ, path formulae ψ, and epistemic formulae κ. State and path formulae of CTL are state and path formulae of PCTLK. Epistemic formulae are expressed using knowledge and group knowledge operators. The syntax of PCTLK is defined as follows:

Definition 4.2. Syntax of PCTLK

Let $p, p_1, p_2, \ldots$ range over the set of atomic propositions Φ_p. Let $A = \{1, \ldots, n\}$ be a set of agents and $G \subseteq A$ be a group of agents, the PCTLK formulae are defined by the following BNF grammar:

$$\phi ::= true \mid p \mid \phi \ \wedge \ \phi \mid \neg\phi \mid \kappa \mid Pr_{\triangledown b}(\psi) \mid Pr_{\triangledown b}(\kappa)$$
$$\psi ::= \bigcirc\phi \mid \phi \ \mathrm{U}^{\leq n} \ \phi \mid \phi \ \mathrm{U} \ \phi$$
$$\kappa ::= K_i\phi \mid E_G\phi \mid C_G\phi \mid D_G\phi$$

where $0 \leq b \leq 1$ is a real number giving the rational boundary, $\triangledown \in \{<, \leq, \geq, >\}$ presents relationship boundary of the probability, and $n \in \mathbb{N}$ is the maximum steps to achieve a specific state.

Formulae κ, called epistemic formulae, are special state formulae in PCTLK that can describe epistemic properties. There are four epistemic modalities: K_i, E_G, C_G, and D_G that represent respectively "agent i knows", "every agent in the group G knows", "common knowledge in the group G", and "distributed knowledge in the group G". $Pr_{\triangledown b}(K_i\phi)$ represents the probability that agent i knows ϕ, where ϕ is a state formula. This probability is $\triangledown b$. $K_i Pr_{\triangledown b}\psi$ states that agent i knows the probability that the path formula ψ holds, which is $\triangledown b$. $\triangledown b$ indicates a lower or upper bound on the probability (e.g $< b$, $\leq b$, $\geq b$, or $> b$). The difference between these two kinds of formulae is that $Pr_{\triangledown b}(K_i\phi)$ expresses the degree of agent i knowing something, while $K_i Pr_{\triangledown b}\psi$ indicates that agent i knows some uncertain things. To

illustrate, $Pr_{\geq 0.9}(K_1\phi)$ indicates that agent 1 knows ϕ with at least 0.9 probability. $K_1(Pr_{\geq 0.9}\phi)$ means agent 1 knows that with at least 0.9 probability, ϕ holds. $\Diamond\phi$ and $\Box\phi$ are the usual abbreviations for eventually and globally: $\Diamond\phi \equiv true$ U ϕ and $\Box\phi \equiv \neg\Diamond\neg\phi$.

There are no universal ($\forall$) and existential ($\exists$) path quantifiers in PCTLK. Instead, the linear temporal operators $\bigcirc$ (next), U (until), and $\mathrm{U}^{\leq n}$ (bounded until) are required to follow the probabilistic operator $Pr_{\triangledown b}$ immediately. The propositional temporal fragment of PCTLK has the same meaning as in CTL. For example, the formula $\bigcirc\phi$ has the meaning of "in the next state ϕ holds". $\phi_1\mathrm{U}\phi_2$ means "ϕ_1 holds until ϕ_2". A new step-bounded variant of until ($\phi_1\mathrm{U}^{\leq n}\phi_2$) is added, meaning that "ϕ_2 will hold within at most n steps while ϕ_1 holds in all states before a ϕ_2-state has been reached". The step-bounded until is necessary in probabilistic logic because the probability of reaching a ϕ_2-state after at most n steps is different from reaching this state after at most $(n+1)$ steps. In CTL, temporal operators $\bigcirc$ and U are required to be immediately preceded by a path quantifier, while in PCTLK, they must follow the operator $Pr_{\triangledown b}$ immediately.

The probabilistic operator on path formulae $Pr_{\triangledown b}(\psi)$ expresses that "ψ holds with a probability $\triangledown b$". For instance, $Pr_{\geq 0.75}(\bigcirc message\_receive)$ asserts that "with at least 0.75 probability, in the next state the message will be received". The probabilistic operator on epistemic formulae $Pr_{\triangledown b}(\kappa)$ states the degree of the knowledge: how much the agent is confident about his knowledge. For example, the following formula: $Pr_{\leq 0.8}(K_1(agent\_2\_has\_resource\_A))$ expresses that agent 1 knows with a maximum probability of 0.8 that agent 2 has resource A. More specifically, this means that out of X accessible states for agent 1, Y states satisfy the fact that agent 2 has resource A, where $Y/X \leq 0.8$.

4.2.2 PCTLK Semantics for DTMC

Before we give the formal semantics of PCTLK, we briefly review the notion of probability space [40] and then define group epistemic relations. A probability space is a mathematical constructor in probability theory. It is expressed as a triple $(\Omega, \mathcal{E}, \mathcal{P})$ that models a process consisting of events that occur randomly.

- Ω is a sample space. To fit our case, we assume that Ω is a set of states.
- $\mathcal{E}$ is a set of events. $\mathcal{E}$ is a subset of Ω. If $\mathcal{E} \subseteq \Omega$, then $\overline{\mathcal{E}} = \Omega - \mathcal{E} \subseteq \Omega$;
- $\mathcal{P} : \mathcal{E} \rightarrow [0, 1]$ is a function, also called a probability measure that assigns probabilities to events. The measure of the whole sample space $\mathcal{P}(\Omega) = 1$. If $\mathcal{E}_1 \subseteq \Omega$, $\mathcal{E}_2 \subseteq \Omega$, and $\mathcal{E}_1 \cap \mathcal{E}_2 = \emptyset$, then $\mathcal{P}(\mathcal{E}_1 \cup \mathcal{E}_2) = \mathcal{P}(\mathcal{E}_1) + \mathcal{P}(\mathcal{E}_2)$.

The probabilistic operator $Pr_{\nabla b}$ can be placed either in front of a path formula ψ or in front of an epistemic formula κ. Formulae are evaluated on states or along paths where a path is an infinite sequence of states. For the formula $Pr_{\nabla b}(\psi)$, $s \models Pr_{\nabla b}(\psi)$ means that "the probability from state s that ψ holds for all outgoing paths is ∇b". For example, $s \models Pr_{<0.25}(true \cup^{\leq 5} \phi)$ asserts that "the probability that the system model satisfies ϕ within 5 steps of all outgoing paths from the state s is less than 0.25". We use the symbol *Prob* to stand for the probability measure. In order to compute the probability measure of a path, we need to associate a probability space with probabilities in the model M_{PIS}. Let $\widehat{\pi} = s_0 \ldots s_m$ be a finite fragment of the path π, the *cylinder set* $Cyl(\widehat{\pi})$ [6] is the set of all infinite reachable paths emanating from $\widehat{\pi}$ in M_{PIS}. The probability measure of this *cylinder set* $Cyl(\widehat{\pi})$ can be calculated by:

$$Prob(Cyl(\widehat{\pi})) = Prob(Cyl(s_0 \ldots s_m)) = I_{init}(s_0) \cdot \mathbf{P_t}(s_0 \ldots s_m) \tag{4.2}$$

where:

$$\mathbf{P_t}(s_0 \dots s_m) = \prod_{0 \leq i < m} \mathbf{P_t}(s_i, s_{i+1}) \tag{4.3}$$

For model checking purposes, Equation 4.2 will be used when we are interested in determining if the model satisfies a given formula. However, when the question is about determining if a given state s satisfies a given formula, we assume that s is the unique initial state, so that we use I_s instead of I_{init} to compute $Prob(Cyl(\widehat{\pi}))$. The I_s value is defined as follows:

$$I_s(t) = \begin{cases} 1, & \text{if } s = t \\ 0, & \text{otherwise} \end{cases} \tag{4.4}$$

Let $G \subseteq A$ be a group of agents. To define the semantics of the epistemic operators E_G, C_G, and D_G, we define the group epistemic accessibility relations from the accessibility relation $\sim_i$ as follows:

Definition 4.3. Group Epistemic Accessibility Relations

- $\sim_G^E$ *is the union of group* G*'s accessibility relations:* $\sim_G^E = \bigcup_{i \in G} \sim_i$.
- $\sim_G^C$ *is the transitive closure of* $\sim_G^E$.
- $\sim_G^D$ *is the intersection of* G*'s accessibility relations:* $\sim_G^D = \bigcap_{i \in G} \sim_i$.

For the state formula $Pr_{\nabla b}(\kappa)$, $s \models Pr_{\nabla b}(\kappa)$ means that "on state s, the probability that the epistemic formula κ holds is ∇b ". We denote the number of states s' such that for a given state s we have $s \sim_i s'$ for agent i ($i \in A$) by $|s \sim_i s'|$. The sample space of agent i at state s is the set of possible worlds or equivalent states of i at s and is equal to $|s \sim_i s'|$. Similarly, we denote the number of states s' that are accessible from a given state s through $\sim_G^E$ by $|s \sim_G^E s'|$, through $\sim_G^C$ by $|s \sim_G^C s'|$, and

through $\sim_G^D$ by $|s \sim_G^D s'|$. We also define $|s \models \phi|$ as follows:

$$|s \models \phi| = \begin{cases} 1, & \text{if } s \models \phi \\ 0, & \text{otherwise} \end{cases} \tag{4.5}$$

Let $s \in W$ be a state, $\pi = s_0, s_1, s_2, \ldots$ a path, i.e. an infinite sequence of states related by transitions, $a \in AP$ an atomic proposition, ϕ a PCTLK state formula (i.e. evaluated over states), and ψ a PCTLK path formula (i.e. evaluated through paths). The $(i+1)th$ state in π is denoted by $\pi(i)$ (i.e., $\pi(i) = s_i$). $\sigma(s)$ is the set of all paths emanating from s. Given the model $M_{PIS} = (W, \mathbf{P_t}, I_{init}, \sim_1, \ldots, \sim_n, V)$, $(M_{PIS}, s) \models \phi$ stands for "state s satisfies ϕ in the system model M_{PIS}" or "ϕ is true at state s in the system model M_{PIS}". $(M_{PIS}, \pi) \models \psi$ is read as "the path π satisfies ψ in the system model M_{PIS}" or "ψ is true through the path π in the system model M_{PIS}". If M_{PIS} is clear from the context, we simply write $s \models \phi$ and $\pi \models \psi$. In the following, we define the semantics of PCTLK.

- For a state s:

$$\begin{aligned} s &\models a && \textit{iff } a \in V(s) \\ s &\models \phi_1 \wedge \phi_2 && \textit{iff } s \models \phi_1 \text{ and } s \models \phi_2 \\ s &\models \neg\phi && \textit{iff } s \not\models \phi \end{aligned}$$

  The semantics of the state formulae $Pr_{\triangledown b}(\psi)$ and $Pr_{\triangledown b}(\kappa)$ will be given later.

- For a path π:

$$\begin{aligned} \pi &\models \bigcirc\phi && \textit{iff } \pi(1) \models \phi \\ \pi &\models \phi_1 \mathrm{U}^{\leq n} \phi_2 && \textit{iff } \exists 0 \leq k \leq n,\ \pi(k) \models \phi_2 \ \textit{ and } \ \forall 0 \leq i < k \ \ \pi(i) \models \phi_1 \\ \pi &\models \phi_1\ \mathrm{U}\phi_2 && \textit{iff } \exists k \geq 0,\ \ \pi(k) \models \phi_2 \ \textit{ and } \ \forall 0 \leq i < k \ \ \pi(i) \models \phi_1 \end{aligned}$$

- The semantics of epistemic formulae κ over a state s is based on the epistemic accessibility relation and group epistemic accessibility relations as **given** in Definition 4.3:

$$
\begin{aligned}
s \models K_i\phi \qquad & \textit{iff } \forall s' \in W \textit{ if } s \sim_i s' \textit{ then } s' \models \phi \\
s \models E_G\phi \qquad & \textit{iff } \forall s' \in W \textit{ if } s \sim_G^E s' \textit{ then } s' \models \phi \\
s \models C_G\phi \qquad & \textit{iff } \forall s' \in W \textit{ if } s \sim_G^C s' \textit{ then } s' \models \phi \\
s \models D_G\phi \qquad & \textit{iff } \forall s' \in W \textit{ if } s \sim_G^D s' \textit{ then } s' \models \phi
\end{aligned}
$$

- In terms of probability space, the set of all reachable states from the initial states is our sample space Ω and for each formula, the set of states satisfying it is the set of events $\mathcal{E}$. Based on this observation, we define the semantics of the probabilistic operator Pr that works on path and epistemic formulae in the following.

  - For a probabilistic operator working on a path formula, where $\widehat{\pi} = s_0 \dots s_m$:

$$
s \models Pr_{\nabla b}(\bigcirc\phi) \qquad \textit{iff} \quad Prob(s, \sigma(s), \bigcirc\phi) \ \nabla b, \qquad \text{where:}
$$

$$
Prob(s, \sigma(s), \bigcirc\phi) = \sum_{\substack{\pi \in \sigma(s) s.t. \\ \pi(1) \models \phi}} \mathbf{P_t}(s, \pi(1)) \tag{4.6}
$$

$$
s \models Pr_{\nabla b}(\phi_1 \mathrm{U}^{\leq n} \phi_2) \qquad \textit{iff} \quad Prob(s, \sigma(s), \phi_1 \mathrm{U}^{\leq n} \phi_2) \ \nabla b, \quad \text{where:}
$$

$$
Prob(s, \sigma(s), \phi_1 \mathrm{U}^{\leq n} \phi_2) = \begin{cases} 1, & \text{if } s \models \phi_2 \\ \displaystyle\sum_{\substack{\forall \pi \in \sigma(s) s.t. \\ \pi \models \phi_1 \mathrm{U}^{\leq n} \phi_2}} Prob(Cyl(\widehat{\pi})), & \text{if } \substack{s_m \models \phi_2,\ 0 < m \leq n, \\ \text{and } \forall 0 \leq i < m,\ s_i \models \phi_1} \\ 0, & \text{otherwise} \end{cases} \tag{4.7}
$$

$$
s \models Pr_{\nabla b}(\phi_1 \mathrm{U} \phi_2) \qquad \textit{iff} \quad Prob(s, \sigma(s), \phi_1 \mathrm{U} \phi_2) \ \nabla b, \qquad \text{where:}
$$

$$Prob(s, \sigma(s), \phi_1 \mathrm{U} \phi_2) = \begin{cases} 1, & \text{if } s \models \phi_2 \\ \sum_{\substack{\forall \pi \in \sigma(s) s.t. \\ \pi \models \phi_1 \mathrm{U} \phi_2}} Prob(Cyl(\widehat{\pi})), & \text{if } \substack{s_m \models \phi_2 \text{ and} \\ \forall 0 \leq i < m,\ s_i \models \phi_1} \\ 0, & \text{otherwise} \end{cases} \tag{4.8}$$

– For a probabilistic operator working on an epistemic formula:

$$s \models Pr_{\triangledown b}(K_i \phi) \quad \textit{iff} \quad Prob(s \models K_i \phi)\ \triangledown b, \quad \text{where:}$$

$$Prob(s \models K_i \phi) = \frac{\sum_{s \sim_i s'} |s' \models \phi|}{|s \sim_i s'|} \tag{4.9}$$

$$s \models Pr_{\triangledown b}(E_G \phi) \quad \textit{iff} \quad Prob(s \models E_G \phi)\ \triangledown b, \quad \text{where:}$$

$$Prob(s \models E_G \phi) \ = \frac{\sum_{s \sim_G^E s'} |s' \models \phi|}{|s \sim_G^E s'|} \tag{4.10}$$

$$s \models Pr_{\triangledown b}(C_G \phi) \quad \textit{iff} \quad Prob(s \models C_G \phi)\ \triangledown b, \quad \text{where:}$$

$$Prob(s \models C_G \phi) = \frac{\sum_{s \sim_G^C s'} |s' \models \phi|}{|s \sim_G^C s'|} \tag{4.11}$$

$$s \models Pr_{\triangledown b}(D_G \phi) \quad \textit{iff} \quad Prob(s \models D_\kappa \phi)\ \triangledown b, \quad \text{where:}$$

$$Prob(s \models D_G \phi) = \frac{\sum_{s \sim_G^D s'} |s' \models \phi|}{|s \sim_G^D s'|} \tag{4.12}$$

Example 4.1. *In this example, we will illustrate how to check if a state satisfies a probabilistic formula using Figure 4.2 that shows a simple M_{PIS} model. The solid lines with numbers are probabilistic transitions. The dash lines are epistemic accessibility relations $\sim_1$ for agent 1. To keep the example simple, only $\sim_1$ from s_0 are shown in the figure.* $\mathbf{P_t}$ *and* I_{init} *are as follows:*

$$\mathbf{P_t} = \begin{bmatrix} 1/2 & 1/2 & 0 \\ 0 & 1/4 & 3/4 \\ 0 & 0 & 1 \end{bmatrix}, \quad I_{init} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

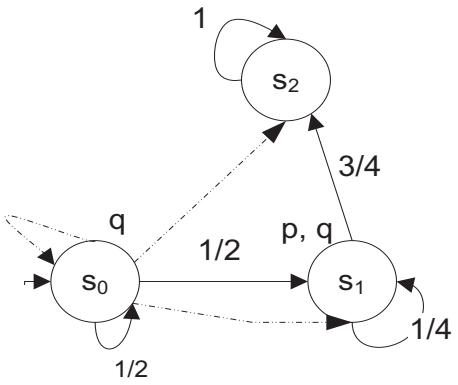

Figure 4.2: An Example of M_{PIS} Model

We want to check if state s_0 satisfies the formula $Pr_{>0.9}(\Diamond p)$. To do so, we have to compute the probability of all paths from s_0 that satisfy $\Diamond p \equiv true$ U p, and then check if the summation of these probabilities is greater than 0.9 (see Equations 4.2, 4.3, and 4.8). We have:

$$\begin{aligned} Prob(s_0, \sigma(s_0), true \text{ U } p) &= \mathbf{P_t}(s_0, s_1) + \mathbf{P_t}(s_0, s_0) \times \mathbf{P_t}(s_0, s_1) \\ &\quad + \mathbf{P_t}(s_0, s_0) \times \mathbf{P_t}(s_0, s_0) \times \mathbf{P_t}(s_0, s_1) + \ldots \\ &= \sum_{n=1}^{\propto} (1/2)^n = 1 > 0.9 \end{aligned}$$

Consequently, state s_0 satisfies this formula.

Let us now check, for example, if s_0 satisfies $Pr_{>0.7}(K_1 q)$. We use Equation 4.9

to compute the probability $Prob(s_0 \models K_1 q)$. *Since the number of accessible states from* s_0 *using* $\sim_1$ *is 3, out of which 2 satisfy* q, *we obtain:*

$$Prob(s_0 \models K_1 q) = 2/3 < 0.7$$

Thus, state s_0 *does not satisfy the formula.*

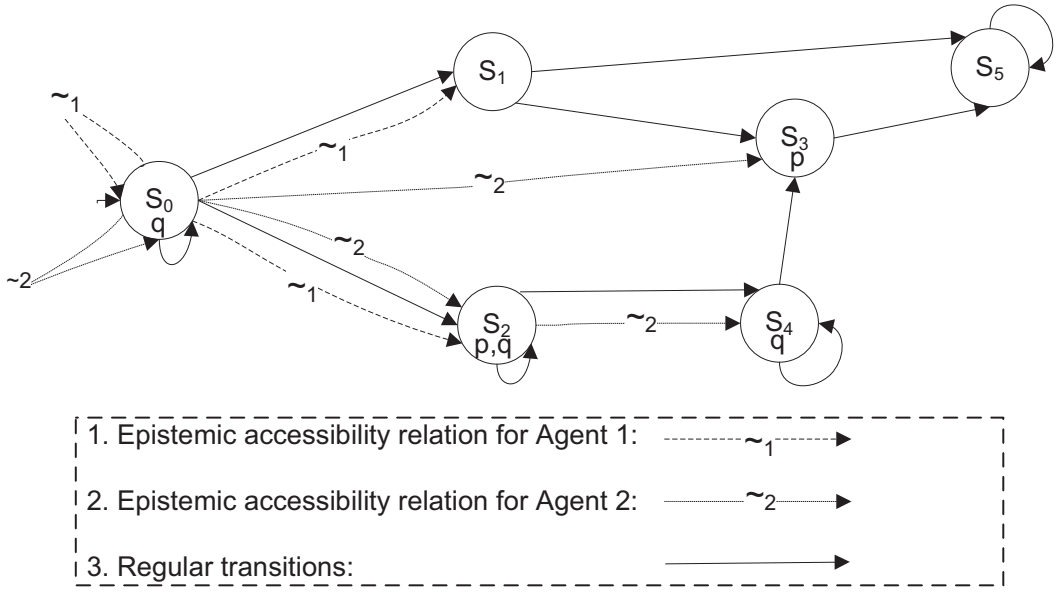

Figure 4.3: Another Example of M_{PIS} *Model*

Example 4.2. *Let us consider another example mainly for probabilistic epistemic operators using Figure 4.3 that illustrates another* M_{PIS} *model where two agents are considered. For the sake of simplicity, only the accessibility relations from state* s_0 *are shown in the figure and one from* s_2 *needed to show* $\sim_G^C$. *Furthermore, the ones that can be deduced from the properties of the accessibility relation (for instance transitivity and Euclideanity) are omitted. We also omit the probabilities of transitions. Let* G *be the group of agents 1 and 2. So, for the union of* $\sim_1$ *and* $\sim_2$ *we have:* $\{(s_0, s_0), (s_0, s_1), (s_0, s_2), (s_0, s_3)\} \subseteq \sim_G^E$; *for the transitive closure of* $\sim_G^E$, *we have:* $\{(s_0, s_0), (s_0, s_1), (s_0, s_2), (s_0, s_3), (s_0, s_4)\} \subseteq \sim_G^C$; *and for the intersection of* $\sim_1$ *and*

$\sim_2$ *we have:* $\{(s_0, s_0), (s_0, s_2)\} \subseteq \sim_G^D$.

Now, we use semantics equations to calculate the probability of epistemic operators.

1. $Prob(s_0 \models K_2 q) = \frac{|s_0 \models q| + |s_2 \models q| + |s_3 \models q|}{3} = \frac{1+1+0}{3} = \frac{2}{3}$
2. $Prob(s_0 \models K_1 q) = \frac{|s_0 \models q| + |s_1 \models q| + |s_2 \models q|}{3} = \frac{1+0+1}{3} = \frac{2}{3}$
3. $Prob(s_0 \models E_G q) = \frac{|s_0 \models q| + |s_1 \models q| + |s_2 \models q| + |s_3 \models q|}{4} = \frac{1+0+1+0}{4} = \frac{2}{4} = \frac{1}{2}$
4. $Prob(s_0 \models D_G q) = \frac{|s_0 \models q| + |s_2 \models q|}{2} = \frac{1+1}{2} = 1$
5. $Prob(s_0 \models C_G q) = \frac{|s_0 \models q| + |s_1 \models q| + |s_2 \models q| + |s_3 \models q| + |s_4 \models q|}{5} = \frac{1+0+1+0+1}{5} = \frac{3}{5}$

Thus, for instance, we have s_0 *satisfies the formula* $Pr_{\geq 0.6}(C_G q)$*, but does not satisfy* $Pr_{\geq 0.7}(C_G q)$.

4.2.3 Properties of Probabilistic Knowledge

PCTLK inherits epistemic properties from CTLK [74] and probabilistic properties of path formulae from PCTL [6]. We list a number of new properties that are not included in conventional epistemic and probabilistic logics. For the conventional ones, please refer to [6, 45].

There are a number of equivalences between a probabilistic epistemic formula and the conventional knowledge formula. If the probability that an agent knows ϕ is greater than or equal to 1 holds at state s, then the agent knows ϕ holds at the state s. We can expand this validity to everyone's knowledge $E_G\phi$, common knowledge $C_G\phi$, and distributed knowledge $D_G\phi$.

Theorem 4.1. Probabilistic and Epistemic Equivalence

$s \models Pr_{\geq 1}(K_i\phi)$ iff $s \models K_i\phi$ $\quad$ $s \models Pr_{<1}(K_i\phi)$ iff $s \models \neg K_i\phi$

$s \models Pr_{\geq 1}(E_G\phi)$ iff $s \models E_G\phi$ $\quad$ $s \models Pr_{<1}(E_G(\phi))$ iff $s \models \neg E_G\phi$

$s \models Pr_{\geq 1}(C_G\phi)$ iff $s \models C_G\phi$ $\quad$ $s \models Pr_{<1}(C_G\phi)$ iff $s \models \neg C_G\phi$

$s \models Pr_{\geq 1}(D_G\phi)$ iff $s \models D_G\phi$ $\quad$ $s \models Pr_{<1}(D_G\phi)$ iff $s \models \neg D_G\phi$

$s \models Pr_{\leq 0}(K_i\phi)$ iff $s \models K_i\neg\phi$ $\quad$ $s \models Pr_{\leq 0}(E_G\phi)$ iff $s \models E_G\neg\phi$

$s \models Pr_{\leq 0}(C_G\phi)$ iff $s \models C_G\neg\phi$ $\quad$ $s \models Pr_{\leq 0}(D_G\phi)$ iff $s \models D_G\neg\phi$

Proof.

We prove the first equivalence; the same method can be used to prove the others.

We first prove $\Rightarrow$.

We have: $s \models Pr_{\geq 1}(K_i(\phi))$. According to the semantics, we get:

$Prob(s \models K_i\phi) = \frac{\sum_{s\sim_i s'} |s' \models \phi|}{|s\sim_i s'|} = 1$.

Therefore, $\sum_{s\sim_i s'} |s' \models \phi| = |s \sim_i s'|$.

This means $\forall s'$ such that $s \sim_i s'$, $s' \models \phi$. Thus, $s \models K_i\phi$.

Next we prove $\Leftarrow$.

$s \models K_i\phi$ iff $\forall s'$ such that $s \sim_i s'$, we have $s' \models \phi$.

Consequently, $\sum_{s\sim_i s'} |s' \models \phi| = |s \sim_i s'|$ Therefore, $s \models Pr_{\geq 1}(K_i(\phi))$ $\square$

Theorem 4.2. Probabilistic and Non-Probabilistic Knowledge

Let b_1 and b_2 be two boundaries in $[0, 1]$. The following validity holds:

$$Pr_{>b_1}(K_i\phi) \wedge Pr_{<b_2}(K_i\phi) \;\Rightarrow\; \neg K_i\phi \wedge \neg K_i\neg\phi$$

Proof. From the left side of the validity, we conclude that $0 \leq b_1 < b_2 \leq 1$. Let s be a state such that: $s \models Pr_{>b_1}(K_i\phi) \wedge Pr_{<b_2}(K_i\phi)$. So we have: $0 < Prob(s \models K_i\phi) = \frac{\sum_{s\sim_i s'} |s' \models \phi|}{|s\sim_i s'|} < 1$. Thus, $0 < \sum_{s\sim_i s'} |s' \models \phi| < |s \sim_i s'|$. This means some accessible states from s satisfy ϕ, but not all of them, and so some others satisfy $\neg\phi$ but not all of them, so the result flows from the semantics of $K_i\phi$. $\square$

Theorem 4.3. Probabilities of Subgroup

1. *If* $s \models E_G\phi$ *and* $G' \subseteq G$, *then* $s \models E_{G'}\phi$. *The same result holds for* C_G *but not for* D_G.

2. *If* $s \models Pr_{\triangledown b}(E_G\phi)$ *and* $G' \subset G$, *then it is not case that* $s \models Pr_{\triangledown b}(E_{G'}\phi)$. *The same result holds for* C_G *and* D_G.

Proof.

- For 1, we prove the theorem for E_G, the same idea can be used for C_G. Let s be a state such that $s \models E_G\phi$. By semantics: $\forall s'$ s.t. $s \sim_G^E s'$, we have $s' \models \phi$. Because $\sim_G^E = \bigcup_{i\in G} \sim_i$, we obtain: $\forall i \in G\ \forall s'$ s.t. $s \sim_i s'$ we have $s' \models \phi$, which also holds for any subset G' of G, so we are done. However, for D_G as the intersection of $\sim_i$ is considered, it is easy to imagine a scenario where a subgroup G' outside the whole intersection so that $D_{G'}$ does not satisfy ϕ.

- For 2, we prove the theorem for E_G; C_G and D_G can be proved similarly. The proof is done by providing two examples with different conclusions. Assume that $(s \models Pr_{\triangledown b}(E_G\phi))$ and $G' \subset G$. Suppose there are four agents in a group: $G = \{Ag_1, Ag_2, Ag_3, Ag_4\}$ and each agent has only two epistemic accessible states, which, except the state itself, are all different. Three agents Ag_1, Ag_2, and Ag_3 at state s know ϕ: $s \models K_1\phi$, $s \models K_2\phi$, and $s \models K_3\phi$. But $s \not\models K_4\phi$. Therefore, $|s \sim_G^E s'| = 5$ and $s \models Pr_{\geq 0.8}(E_G\phi)$ because $Prob(s \models E_G\phi) = \frac{4}{5} = 0.8$. There is a subgroup $G1 = \{Ag_1, Ag_2, Ag_3\} \subset G$ where $s \models Pr_{\geq 0.8}(E_{G1}\phi)$ because $|s \sim_{G1}^E s'| = 4$, so $Prob(s \models E_{G1'}\phi) = \frac{4}{4} = 1 \geq 0.8$, while for another subgroup $G2 = \{Ag_1, Ag_2, Ag_4\} \subset G$, $s \not\models Pr_{\geq 0.8}(E_{G2}\phi)$ because $|s \sim_{G2}^E s'| = 3$ and $Prob(s \models E_{G2}\phi) = \frac{3}{4} = 0.75 < 0.8$.

□

Theorem 4.4. Extended Properties

For all formulas ϕ and ψ, and all agents $i = 1, \ldots, n$, the following extended properties for probability hold:

1. $K_i(Pr_{\nabla b}K_i\phi) \Rightarrow K_i(K_i(Pr_{\nabla b}K_i\phi))$
2. $Pr_{\nabla b}K_i\phi \wedge K_i(\phi \Rightarrow \varphi) \Rightarrow Pr_{\nabla b}K_i\varphi$
3. $K_i(Pr_{\nabla b}K_i\phi) \Rightarrow Pr_{\nabla b}K_i\phi$

Proof.

- 1 follows directly from the transitivity of $\sim_i$.
- For 2, if $s \models Pr_{\nabla b}K_i\phi \wedge K_i(\phi \Rightarrow \varphi)$, then for all states s' such that $(s \sim_i s')$ we have $s' \models (\phi \Rightarrow \varphi)$ and $\frac{\sum_{s\sim_i s'} |s' \models \phi|}{|s\sim_i s'|} \nabla b$. It follows, using modus ponens, that $\sum_{s\sim_i s'} |s' \models \varphi| = \sum_{s\sim_i s'} |s' \models \phi|$. Consequently $\frac{\sum_{s\sim_i s'} |s' \models \varphi|}{|s\sim_i s'|} \nabla b$, so the results.
- 3 follows directly from the reflexivity of $\sim_i$.

□

4.3 Model Checking Technique

We propose a reduction-based approach to transform the problem of model checking PCTLK into the problem of model checking PCTL so that the PCTL's model checker, PRISM, can be used to verify PCTLK formulae. Given a PCTLK model M_{PIS} and a PCTLK formula φ_{PIS}, we can define a PCTL model $M = \mathcal{F}(M_{PIS})$ and a PCTL formula $\varphi = \mathcal{F}(\varphi_{PIS})$ such that $(M_{PIS}, s_0) \models \varphi_{PIS}$ iff $(\mathcal{F}(M_{PIS}), s_0) \models \mathcal{F}(\varphi_{PIS})$. Thus, the reduction is implemented in two parts: (1) transforming the probabilistic epistemic model; and (2) reducing PCTLK formulae.

The workflow for model checking PCTLK is summarized in Figure 4.4. The transformation $\mathcal{F}$ translates the inputs: probabilistic interpreted system M_{PIS} and PCTLK formula φ_{PIS} into a regular DTMC model $\mathcal{F}(M_{PIS})$ and PCTL formula $\mathcal{F}(\varphi_{PIS})$. This transformation is done automatically by a tool we have implemented, then we use the PRISM model checker to verify if the obtained model satisfies the obtained formula. In the following, we split the explanation into two parts: models transformation and formulae reduction to introduce the details of our model checking approach.

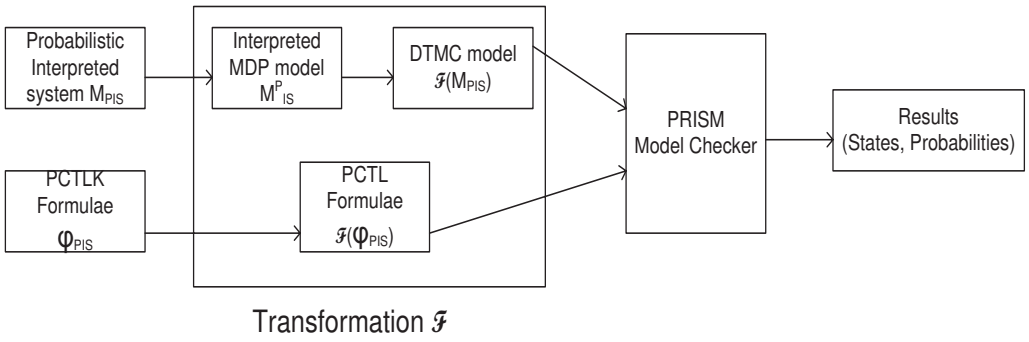

Figure 4.4: Verification Workflow for PCTLK

4.3.1 Translation of M_{PIS} Models

The translation from an M_{PIS} model, which is an epistemic DTMC model, to an $\mathcal{F}(M_{PIS})$ model is done in two steps. In the first step we translate the probabilistic epistemic model M_{PIS} into an equivalent interpreted MDP model M^P_{IS} that will be formally defined later (see Figure 4.4). In this step, the key operation is the mapping of the epistemic relations $(\sim_i, \sim^E_G, \sim^C_G, \sim^D_G)$ used in M_{PIS} to specific actions $(ACC_i, ACC^E_G, ACC^C_G, ACC^D_G)$ needed for M^P_{IS}. Then, in the second step we transform this equivalent MDP model M^P_{IS} into a regular DTMC model $M = \mathcal{F}(M_{PIS})$ by selecting the specific action for the formula. These two steps will be explained in this

section.

The motivation behind the first step is that an M_{PIS} is a DTMC model, which therefore cannot model the nondeterministic behavior of the concurrent processes in an adequate manner because it only allows deterministic choices. However, Markov Decision Processes (MDP) that can be viewed as a variant of Markov chains extending DTMC by allowing nondeterministic choices. Thus, both nondeterministic and probabilistic choices coexist in MDP. The definition of MDP as given in [6] is as follows.

Definition 4.4. MDP

Over a set of atomic propositions AP, an MDP can be expressed as a tuple $(S,\ Act,\ \mathbf{P}\ ,\ I_{init},\ L,\ AP)$*, where:*

- S *is a nonempty and finite set of states.*
- $\mathbf{P} : S \times Act \times S \to [0,1]$ *is the transition probability function, such that for every state* $s \in S$ *and action* $\alpha \in Act$*, we have* $\sum_{s' \in S} \mathbf{P}(s,\ \alpha,\ s') \in \{0,1\}$.
- Act *is a set of actions. At the state* $s \in S$*, the action* α *is enabled iff* $\sum_{s' \in S} \mathbf{P}(s,\ \alpha,\ s') = 1$.
- $I_{init} : S \to [0,1]$ *is the initial distribution such that for all states* $s \in S$*,* $\sum_{s \in S} I_{init}(s) = 1$.
- $L : S \to 2^{AP}$ *is a state labeling function.*

For any state $s \in S$, there is at least one enabled action α. The operational behavior of an MDP starts with a state s_0 such that $I_{init}(s_0) > 0$. At every state s, the system first chooses an enabled action nondeterministically from the set $Act(s)$ of enabled actions at s. Then, it performs this action probabilistically according to the transition probability function. Thus, a DTMC is a special MDP in which for

any state s, there is only one action in $Act(s)$. Therefore, by adding actions into a DTMC, we can create an MDP. Now we can define the M^P_{IS} MDP model obtained from our M_{PIS} model (step 1).

Definition 4.5. M^P_{IS} **Model**

Given an epistemic DTMC model $M_{PIS} = (W, \mathbf{P_t}, I_{init}, \sim_1, \ldots, \sim_n, V)$,
$M^P_{IS} = (S,\ Act^P,\ \mathbf{P}^P_t,\ I^P_{init},\ V^P)$ *is an MDP model defined from* M_{PIS} *as follows:*

- $S = W$.
- $Act^P = \{Run, Acc_1, \ldots, Acc_n, Acc^E_G, Acc^C_G, Acc^D_G\}$ *is a set of actions. The action Run labels the transitions obtained from the* M_{PIS} *transitions, while each action* Acc_i *labels the transitions obtained from the epistemic accessibility relation* $\sim_i$. *The actions* Acc^E_G, Acc^C_G, *and* Acc^D_G *label the transitions obtained from the accessibility relations* $\sim^E_G$, $\sim^C_G$, *and* $\sim^D_G$ *respectively.*
- $\mathbf{P}^P_t : S \times Act^P \times S \to [0,1]$ *is the transition probability function defined as follows: for all* $s \in S$,

$$\mathbf{P}^P_t(s,\alpha,s') = \begin{cases} \mathbf{P}_t(s,s'), & \text{if } \alpha = Run \\ \frac{1}{|s\sim_i s'|}, & \text{if } \alpha = Acc_i \\ \frac{1}{|s\sim^E_G s'|}, & \text{if } \alpha = Acc^E_G \\ \frac{1}{|s\sim^C_G s'|}, & \text{if } \alpha = Acc^C_G \\ \frac{1}{|s\sim^D_G s'|}, & \text{if } \alpha = Acc^D_G \end{cases}$$

- $I^P_{init} = I_{init}$.
- $V^P = V$.

According to this translation, the transitions of M^P_{IS} are obtained (1) from the transitions of M_{PIS} with the same probabilities; and (2) from the epistemic relations where the probabilities are equally distributed depending on the number of accessible states from each given state. For example, if 3 states are accessible from a state s, then each of the the three obtained transitions will have 1/3 as probability.

Because M^P_{IS}, as an MDP, is not deterministic, a *policy* or *strategy* should be used to resolve all the nondeterministic choices by picking an enabled transition for a state, which induces a Markov chain. In the literature related to MDP, this policy is called *scheduler* (or also *adversary*). A *scheduler* is a function from the state set S to the action set Act^P such that it chooses in any state s one of the enabled actions. We define five particular schedulers $\lambda, \lambda_i, \lambda^E_G, \lambda^C_G$, and λ^D_G that are used to obtain a DTMC from the obtained MDP model M^P_{IS} (step 2).

From each state in M^P_{IS}, the following actions are enabled: *Run*, Acc_i, Acc^E_G, Acc^C_G, and Acc^D_G. We set the scheduler λ that always selects the transitions labeled by *Run* (so the transitions obtained by the accessibility relations are ignored). The scheduler λ_i always selects the action Acc_i from a state s and then selects the action *Run* from all the following states (so first the accessibility relations $\sim_i$ are considered and then the normal transitions). Similarly, the schedulers λ^E_G, λ^C_G, and λ^D_G always select the actions Acc^E_G, Acc^C_G, and Acc^D_G respectively from a state s and then select the action *Run* from all the following states. It is easy to see that the obtained models are DTMC for PCTL, which means, the models $M = \mathcal{F}(M_{PIS})$ as explained above.

The following example illustrates this transformation procedure. Figure 4.5 (a) is the MDP model for a system with two agents ($i \in \{1, 2\}$). The two agents have the same two local states: $L_i = \{s_1,\ s_2\}$. Thus four combinations of those local states are possible, making the number of possible global states equal to four: $\{g_1, g_2, g_3, g_4\}$, where $g_1 = (s_1, s_1)$; $g_2 = (s_1, s_2)$; $g_3 = (s_2, s_1)$; and $g_4 = (s_2, s_2)$. For

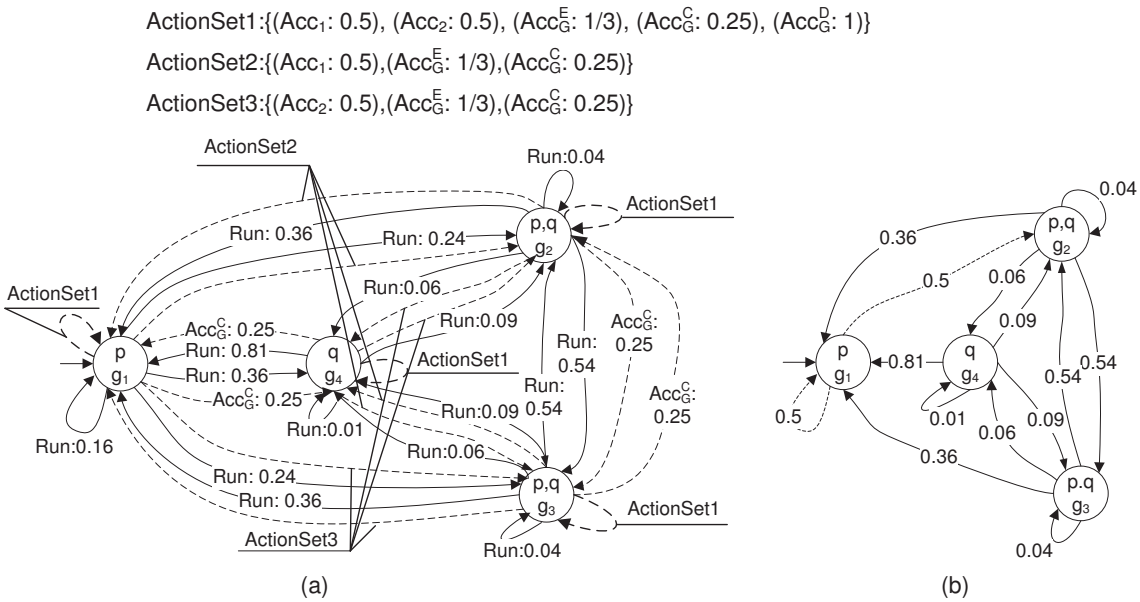

Figure 4.5: (a): MDP M_{IS}^{P} Model. (b): Scheduler λ_1 at State g_1

simplicity, the two agents have the same probabilistic transition function T_i specified as follows: $T_i(s_1, Run, s_1) = 0.4$; $T_i(s_1, Run, s_2) = 0.6$; $T_i(s_2, Run, s_1) = 0.9$; and $T_i(s_2, Run, s_2) = 0.1$. p and q are atomic propositions and the probabilistic transitions are shown in the figure as labeled transitions, where the labels have the form $(action - name : probability - value)$, for instance $(Run : 0.24)$ and $(Acc_1 : 0.5)$. The probabilities associated with the action Run are calculated based on the T_i function. For instance, the probability of the transition (g_1, Run, g_2) is computed as follows: $T_i(s_1, Run, s_1) \times T_i(s_1, Run, s_2) = 0.4 \times 0.6 = 0.24$. The probabilities associated with the action Acc_i are given and are used to compute those associated with the actions Acc_G^E, Acc_G^C, and Acc_G^D. Figure 4.5 (b) is the same system with scheduler λ_1 at the global state g_1. This means, at g_1 the action Acc_1 is chosen and then the action Run is chosen from all the other three global states g_2, g_3, and g_4.

4.3.2 Reducing PCTLK to PCTL

Having transformed the models, now we need to reduce PCTLK formulae into PCTL formulae. PCTL has the same syntax as PCTLK, except that the epistemic formulae κ are not included. This reduction process works in stages inductively. Each PCTLK formula is divided into "maximal state subformulae" such that each maximal state subformula ϕ: (1) includes the probabilistic or epistemic operators; (2) differs from ϕ; and (3) is not contained in any other state subformula of ϕ. Each maximal state subformula is a PCTLK formula, which can be divided into other maximal state subformulae until no new maximal state subformula can be identified. Thus, in stage k, formulae of level k are decomposed into maximal state subformulae of level $k-1$ or lower. The lowest level, level 0, contains only atomic propositions and the highest level is the whole formula. In stage k all maximal state subformulae of ϕ of level smaller than k are processed and replaced by new atomic propositions, which are labels of the subformulae. Before giving an example let us introduce the reduction rules. Let $\phi,\ \phi_1,\ \phi_2$ be PCTLK formulae and a be an atomic proposition, the transformation rules are defined recursively as in Table 4.1:

Table 4.1: PCTLK to PCTL Transformation Rules

Rule Number	Rules
1	$\mathcal{F}(a) = a$;
2	$\mathcal{F}(\neg\phi) = \neg\mathcal{F}(\phi)$;
3	$\mathcal{F}(\phi_1 \wedge \phi_2) = \mathcal{F}(\phi_1) \wedge \mathcal{F}(\phi_2)$;
4	$\mathcal{F}(Pr_{\nabla b} \bigcirc \phi) = Pr_{\nabla b} \bigcirc \mathcal{F}(\phi)$;
5	$\mathcal{F}(Pr_{\nabla b}(\phi_1 \mathrm{U} \phi_2)) = Pr_{\nabla b}(\mathcal{F}(\phi_1) \mathrm{U} \mathcal{F}(\phi_2))$;
6	$\mathcal{F}(Pr_{\nabla b}(\phi_1 \mathrm{U}^{\leq n} \phi_2)) = Pr_{\nabla b}(\mathcal{F}(\phi_1) \mathrm{U}^{\leq n} \mathcal{F}(\phi_2))$;
7	$\mathcal{F}(K_i\phi) = Pr_{\geq 1}(\bigcirc \mathcal{F}(\phi))$;
8	$\mathcal{F}(E_G\phi) = Pr_{\geq 1}(\bigcirc \mathcal{F}(\phi))$;
9	$\mathcal{F}(C_G\phi) = Pr_{\geq 1}(\bigcirc \mathcal{F}(\phi))$;
10	$\mathcal{F}(D_G\phi) = Pr_{\geq 1}(\bigcirc \mathcal{F}(\phi))$;
11	$\mathcal{F}(Pr_{\nabla b} K_i\phi) = Pr_{\nabla b}(\bigcirc \mathcal{F}(\phi))$;
12	$\mathcal{F}(Pr_{\nabla b} E_G\phi) = Pr_{\nabla b}(\bigcirc \mathcal{F}(\phi))$;
13	$\mathcal{F}(Pr_{\nabla b} C_G\phi) = Pr_{\nabla b}(\bigcirc \mathcal{F}(\phi))$;
14	$\mathcal{F}(Pr_{\nabla b} D_G\phi) = Pr_{\nabla b}(\bigcirc \mathcal{F}(\phi))$.

To complete the reduction process, a DTMC model $M = (S, \mathbf{P}, I_{init}, L, AP)$ associated with each PCTL formula should be defined. This is done by specifying which scheduler is associated with which formula. In the following, $(M, s) \models_{Sch} \phi$ means the PCTL formula ϕ is satisfied in the model M obtained by applying the scheduler Sch at state s. The following theorem is a direct consequence of the definition of $\mathcal{F}$ and can be easily proved by induction on the structure of the formula.

Theorem 4.5. Satisfaction Equivalence

$$(M_{PIS}, s) \models a \textit{ iff } (\mathcal{F}(M_{PIS}), s) \models_{\lambda} a$$
$$(M_{PIS}, s) \models \neg\phi \textit{ iff } (\mathcal{F}(M_{PIS}), s) \models_{\lambda} \neg\mathcal{F}(\phi)$$
$$(M_{PIS}, s) \models \phi_1 \wedge \phi_2 \textit{ iff } (\mathcal{F}(M_{PIS}), s) \models_{\lambda} \mathcal{F}(\phi_1) \wedge \mathcal{F}(\phi_2)$$
$$(M_{PIS}, s) \models Pr_{\triangledown b} \bigcirc \phi \textit{ iff } (\mathcal{F}(M_{PIS}), s) \models_{\lambda} Pr_{\triangledown b} \bigcirc \mathcal{F}(\phi)$$
$$(M_{PIS}, s) \models Pr_{\triangledown b}(\phi_1 \mathrm{U} \phi_2) \textit{ iff } (\mathcal{F}(M_{PIS}), s) \models_{\lambda} Pr_{\triangledown b}(\mathcal{F}(\phi_1) \mathrm{U} \mathcal{F}(\phi_2))$$
$$(M_{PIS}, s) \models Pr_{\triangledown b}(\phi_1 \mathrm{U}^{\leq n} \phi_2) \textit{ iff } (\mathcal{F}(M_{PIS}), s) \models_{\lambda} Pr_{\triangledown b}(\mathcal{F}(\phi_1) \mathrm{U}^{\leq n} \mathcal{F}(\phi_2))$$
$$(M_{PIS}, s) \models K_i\phi \textit{ iff } (\mathcal{F}(M_{PIS}), s) \models_{\lambda_i} Pr_{\geq 1}(\bigcirc \mathcal{F}(\phi))$$
$$(M_{PIS}, s) \models E_G\phi \textit{ iff } (\mathcal{F}(M_{PIS}), s) \models_{\lambda_G^E} Pr_{\geq 1}(\bigcirc \mathcal{F}(\phi))$$
$$(M_{PIS}, s) \models C_G\phi \textit{ iff } (\mathcal{F}(M_{PIS}), s) \models_{\lambda_G^C} Pr_{\geq 1}(\bigcirc \mathcal{F}(\phi))$$
$$(M_{PIS}, s) \models D_G\phi \textit{ iff } (\mathcal{F}(M_{PIS}), s) \models_{\lambda_G^D} Pr_{\geq 1}(\bigcirc \mathcal{F}(\phi))$$
$$(M_{PIS}, s) \models Pr_{\triangledown b}(K_i\phi) \textit{ iff } (\mathcal{F}(M_{PIS}), s) \models_{\lambda_i} Pr_{\triangledown b}(\bigcirc \mathcal{F}(\phi))$$
$$(M_{PIS}, s) \models Pr_{\triangledown b}(E_G\phi) \textit{ iff } (\mathcal{F}(M_{PIS}), s) \models_{\lambda_G^E} Pr_{\triangledown b}(\bigcirc \mathcal{F}(\phi))$$
$$(M_{PIS}, s) \models Pr_{\triangledown b}(C_G\phi) \textit{ iff } (\mathcal{F}(M_{PIS}), s) \models_{\lambda_G^C} Pr_{\triangledown b}(\bigcirc \mathcal{F}(\phi))$$
$$(M_{PIS}, s) \models Pr_{\triangledown b}(D_G\phi) \textit{ iff } (\mathcal{F}(M_{PIS}), s) \models_{\lambda_G^D} Pr_{\triangledown b}(\bigcirc \mathcal{F}(\phi))$$

Simply put, this theorem states that if the PCTLK formula does not include epistemic operators, then the corresponding PCTL formula is satisfied in the DTMC obtained by only considering the normal transitions. However, if the formula has the

form of $K_i\phi$, then the corresponding PCTL formula is satisfied in the DTMC obtained by considering first the epistemic accessibility relation $\sim_i$ and then the normal transitions, which shows why the K operator is translated to the next operator. The same intuition holds for the the other epistemic formulae.

Theorem 4.6. Soundness and Completeness

Let M_{PIS} a probabilistic interpreted system and ϕ be a PCTLK formula, and let $\mathcal{F}(M_{PIS})$ and $\mathcal{F}(\phi)$ be the corresponding PCTL model and PCTL formula. Then $M_{PIS} \models \phi$ if and only if $\mathcal{F}(M_{PIS}) \models_x \mathcal{F}(\phi)$, where $x \in \{\lambda, \lambda_i, \lambda_G^E, \lambda_G^C, \lambda_G^D\}$ is a scheduler.

Proof.

We prove this theorem by induction on the structure of the formula ϕ. The rules 1 to 6 are straightforward (see Table 5.1). We need to consider the rules 7 to 14.

- For rule 7: We know that $(M_{PIS}, s) \models K_i\psi$ iff $\forall s' \in W$, if $s \sim_i s'$, then $(M_{PIS}, s') \models \psi$. Therefore, $\forall s' \in S$, with the scheduler λ_i, it follows that every infinite path π emanating from s satisfies that $\pi(1) = s'$ and $(\mathcal{F}(M_{PIS}), s') \models_{\lambda_i} \mathcal{F}(\psi)$. Also, the action for the scheduler λ_i is $\alpha = Acc_i$ and $\mathbf{P}_t^p(s, \alpha, s') = \frac{1}{|s \sim_i s'|}$. By the semantics of $Pr_{\geq 1}\bigcirc$, and $\frac{\sum_1^{|s\sim_i s'|} 1}{|s\sim_i s'|} = 1$, we obtain $(\mathcal{F}(M_{PIS}), s) \models_{\lambda_i} Pr_{\geq 1}(\bigcirc\mathcal{F}(\psi))$, and vice versa.

  Then same method can be used to prove rules 8 to 10 using different scheduler λ_G^E, λ_G^C, and λ_G^D.

- For rule 11: We have that $(M_{PIS}, s) \models Pr_{\triangledown b}K_i\psi$ iff $Prob(s \models K_i\psi) \triangledown b$, where $Prob(s \models K_i\psi) = \frac{\sum_{s\sim_i s'} |s' \models \psi|}{|s\sim_i s'|}$. Recall that the probability transitions: $\mathbf{P}_t^p(s, Acc_i, s') = \frac{1}{|s\sim_i s'|}$ for scheduler λ_i at state s; therefore, $Prob(\pi \in Path^{\lambda_i}(s) : (\mathcal{F}(M_{PIS}, s') \models_{\lambda_i} \mathcal{F}(\psi)) = \sum_{\substack{\pi\in\sigma(s) s.t. \\ \pi(1)\models_{\lambda_i}\mathcal{F}(\psi)}} \mathbf{P}_t^p(s, Acc_i, \pi(1)) = \frac{\sum_{s\sim_i s'} |s' \models_{\lambda_i} \mathcal{F}(\psi)|}{|s\sim_i s'|}$, where $\pi(1) = s'$. By the semantics of $Pr_{\triangledown b}\bigcirc$, we obtain $(\mathcal{F}(M_{PIS}), s) \models_{\lambda_i}$

$Pr_{\nabla b}(\bigcirc\mathcal{F}(\psi))$, and vice versa.

Then same method can be used to prove rules 12 to 14 using different scheduler λ_G^E, λ_G^C, and λ_G^D.

Therefore, the theorem. □

We use the previous example shown in Figure 4.5 to illustrate how to convert PCTLK formulae into PCTL by levels and identify the states that satisfy each maximal state subformula. The PCTLK formula $K_1(Pr_{\geq 0.50}((Pr_{>0.70} \bigcirc p)\mathrm{U}q))$ asserts that agent 1 knows that at least in 50% of the cases, there is more than 70% of chance that in the next state p holds until q. The levels of the subformulae of this formula and the new atomic propositions labeling each maximal state subformula are shown in Table 4.2:

Table 4.2: Example of Reduction of The Formula $K_1(Pr_{\geq 0.50}((Pr_{>0.70} \bigcirc p)\mathrm{U}q))$

Level	Subformulae	Transformed subformulae	Labeled states set
Level 0	p	$\phi_1^0 = p$	$\{g_1,\ g_2,\ g_3\}$
	q	$\phi_2^0 = q$	$\{g_2,\ g_3,\ g_4\}$
Level 1	$Pr_{>0.7} \bigcirc p$	$\phi_1^1 = Pr_{>0.7} \bigcirc \phi_1^0$	$\{g_1,\ g_2,\ g_3,\ g_4\}$
Level 2	$Pr_{\geq 0.5}((Pr_{>0.7} \bigcirc p)\mathrm{U}q)$	$\phi_1^2 = Pr_{\geq 0.5}(\phi_1^1 \mathrm{U} \phi_2^0)$	$\{g_1,\ g_2,\ g_3,\ g_4\}$
Level 3	$K_1(Pr_{\geq 0.5}((Pr_{>0.7} \bigcirc p)\mathrm{U}q))$	$\phi_1^3 = Pr_{\geq 1} \bigcirc \phi_1^2$	$\{g_1,\ g_2,\ g_3,\ g_4\}$

4.4 Case Study

We implement our reduction-based model checking technique on top of PRISM as a tool that takes as input PCTLK formulae and M_{PIS} model and produces as output PCTL formulae and DTMC model $\mathcal{F}(M_{PIS})$, which are the inputs of PRISM. PRISM [66], a probabilistic symbolic model checker, is a tool for formal modeling and analysis of systems which exhibit random or probabilistic behavior. It supports three types of probabilistic models, discrete-time Markov chains (DTMCs), continuous-time Markov chains (CTMCs) and Markov decision processes (MDPs). These models can

be specified in the PRISM modeling language with a simple, state-based language that is based on the Reactive Modules formalism of Alur and Henzinger [3]. The property specification language incorporates the temporal logics $PCTL$, CSL, LTL, and $PCTL^*$.

We apply the approach using the modified protocol of Chaum's dining cryptographers [25] as case study. The Dining Cryptographers protocol (DC) aims to get information from anonymous broadcasting messages and has already been modeled using agents by various authors [56, 60, 73, 94]. We add an uncertainty situation to the original protocol and extend it into the Cheating Dining Cryptographers protocol (CDC). The epistemic and probabilistic properties of the CDC protocol are automatically transformed into PCTL and verified by the PRISM probabilistic model checker.

4.4.1 Protocol Description

Anonymity is an important issue in the field of modern cryptography. The original Dining Cryptographers (DC) Protocol is introduced by the following scenario [25]:

> Three cryptographers are sitting down to dinner at their favorite three-star restaurant. Their waiter informs them that arrangements have been made with the maitre d'hotel for the bill to be paid anonymously. One of the cryptographers might be paying for dinner, or it might have been NSA (U.S. National Security Agency). The three cryptographers respect each others right to make an anonymous payment, but they wonder if NSA is paying.

Based on the assumption that at most one cryptographer is paying, the following rules can solve the dining cryptographers' quandary [25]:

1. Each cryptographer flips an unbiased coin and only shows the outcome to the cryptographer on his right.

2. Each cryptographer states whether the two coins he can see are on the same side or on different sides.

3. The cryptographer who pays the dinner states the opposite of what he sees.

After running this protocol, all the cryptographers can determine whether it was the NSA or one of the cryptographers who paid for dinner: an odd number of differences indicates that a cryptographer is paying; an even number indicates that NSA is paying. Also, if a cryptographer is paying, neither of the other two can figure out who is paying. This protocol can also be applied to more than three cryptographers.

We assume that cryptographers may make mistakes or deliberately break the protocol by certain probability (we use cheating index p to indicate it). We aim to investigate how cheating index p affects the accuracy of cryptographers' inference. Kacprzak et al. in [60] also mentioned Cheating Dining Cryptographers (CDC) protocol but without indicating the degree of cheating. The DC protocol can be seen as a special CDC protocol with cheating index $p = 0$ for all the cryptographers. We discuss the encoding and verification of the CDC protocol in the following section.

4.4.2 Protocol Encoding

Our modeling techniques are completely compatible with PRISM implementation. We translate every agent into a *module* in PRISM and the entire MAS is defined as a system with *agent modules* which are all synchronized. This PRISM system can be viewed as DTMC model scheduled using λ. To implement the epistemic schedulers λ_i, λ_G^E, λ_G^C, and λ_G^C, we use labels to define the epistemic relations. To illustrate, label $A1s1$ includes a set of states that are equivalent for agent $A1$ at s_1 (λ_1), while label

$A1A2s0E$ or $A1A2s0D$ contains the union or joint of group (agent 1 and agent 2) accessibility relations at global state (s_0) respectively.

```
label "A1s1" = (s1=0|s1=1|s1=2);
label "A1A2s0E"=(s1=0|s1=1|s1=2|s1=4|s1=5);
label "A1A2s0D" =(s1=0);
```

Therefore, $A1s1$ is considered as the set of states that comprises all the states related to s_1 by $\sim_1$. Similarly, $A1A2s0E$ and $A1A2s0D$ are the state set related to s_0 by $\sim_G^E$ and $\sim_G^D$.

We model the CDC protocol differently from that presented in [60]. We add cheating index p to indicate the probability of a cryptographer being dishonest (or making mistakes) since probabilistic interpreted systems allow probabilistic transitions. For simplification but without loss of generality, we assume that all the cryptographers have the same cheating index p and generalization to n indices is straightforward. When we set all the cryptographers' cheating indices to 0, the CDC protocol becomes DC protocol.

To formalize the protocol, we naturally assume that NSA and the three cryptographers have an equal chance of paying the bill. We encode the scenario by using a probabilistic interpreted system. Each cryptographer C_i is comprised of two variables: *flipping the coin* variable and *stating results* variable. The variable *flipping the coin* uses 3 states: *flipping coin*, *head, tail*. The variable *stating results* has 9 states with the meaning is intuitively explained by their labels: *NotDecided*, *pay, notPay, seeEqual/pay, seeDiff/pay, seeEqual/notPay, seeDiff/notPay, saidEqual,* and *saidDiff* (see Figure 4.6).

To model the CDC protocol in PRISM, we add a synchronizing flag for every cryptographer agent to avoid deadlock. When a cryptographer has announced the outcome, the synchronizing value is set to 1 and stays. After all the cryptographers'

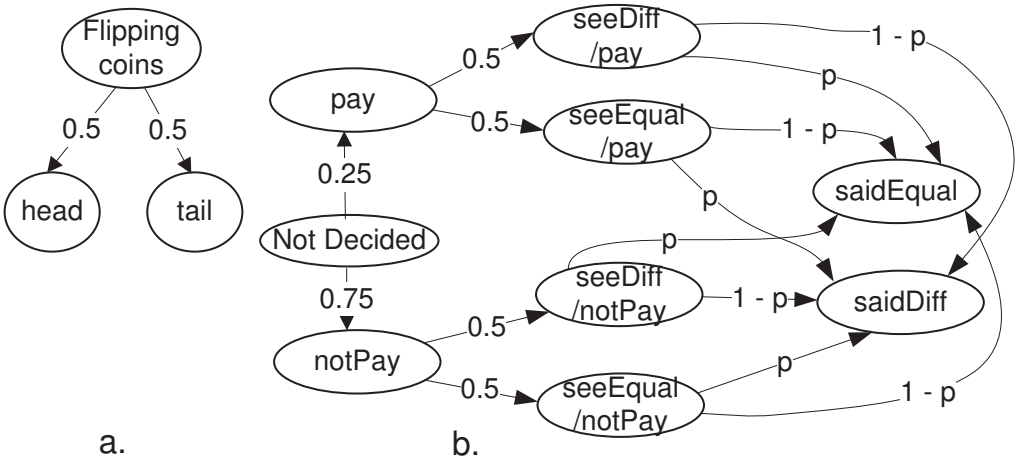

Figure 4.6: Cheating Cryptographers Protocol (a. Flipping Coins, b. Stating Outcomes)

synchronizing flags have been set, the system will count all *saidDiff* and to see if the number of *saidDiff* is even or odd we use the integer modulo operation *mod*.

```
label "even" = fun(mod, (diff1+diff2+diff3),2)=0;
label  "odd" = fun(mod, (diff1+diff2+diff3),2)=1;
```

4.4.3 Experimental Results

We analyze the verification results for the DC and CDC protocols. The presented experimental results were performed on a DELL desktop computer with 1.86 GHz Intel Core Duo T6300 processor and 3.25 GB memory under 32-bit Windows WinXP professional version 2002 Service Pack3 Operating System.

Table 5.4 shows the size and run time for the models that we built for 3 to 10 cryptographers for the DC and CDC protocols. The number of states and transitions are our model size. The construction time is the time for converting the PRISM model into Multi-Terminal Binary Decision Diagram (MTBDD) symbolic model. We can see as the number of cryptographers increases, the model size increases dramatically, but PRISM can still handle the large number of states and transitions thanks to the

Table 4.3: Experimental Results with PRISM

Number of Crypt. (n)	Dining Cryptographer				Cheating Dining Cryptographer			
	Number of States	Number of Transitions	Construction Time (sec.)	Execution Time (sec.)	Number of States	Number of Transitions	Construction Time (sec.)	Execution Time (sec.)
3	261	452	$<$ **0.01**	$<$ **0.01**	503	1,036	0.016	0.017
4	1,286	2,725	0.015	0.015	3,146	8,063	0.016	0.021
5	6,151	15,750	0.031	0.046	18,753	57,882	0.032	0.040
6	28,680	86,919	0.047	0.062	109,378	399,273	0.034	0.049
7	131,081	460,808	0.063	0.078	625,003	2,643,480	0.078	0.109
8	589,834	2,363,913	0.110	0.157	3,515,628	16,936,299	0.094	0.126
9	2,621,451	11,806,730	0.125	0.172	19,531,253	105,670,630	0.141	0.177
10	11,534,348	57,694,219	0.157	0.220	107,421,878	645,191,965	0.188	0.266
11	5.03E+7	2.77E+8	0.128	0.327	5.86E+8	3.87E+9	0.265	0.374
12	2.18E+8	1.31E+9	0.203	0.281	3.26E+9	2.36E+10	0.204	0.297
13	9.40E+8	6.11E+9	0.328	0.515	1.71E+10	1.33E+11	0.453	0.625
14	4.03E+9	2.82E+10	0.469	0.734	9.16E+10	7.69E+11	0.562	0.843
15	1.72E+10	1.29E+11	0.515	0.812	4.88E+11	4.39E+12	0.563	0.860
16	7.30E+10	5.84E+11	0.703	1.047	2.59E+12	2.49E+13	0.765	1.156
17	3.09E+11	2.63E+12	0.813	1.251	1.37E+13	1.40E+14	0.86	1.313
18	1.31E+12	1.18E+13	1.063	1.782	7.25E+13	7.83E+14	1.172	1.797
19	5.50E+12	5.22E+13	1.250	2.047	3.81E+14	4.35E+15	1.515	2.452
20	2.31E+13	2.31E+14	1.546	2.577	2.00E+15	2.40E+16	1.703	2.719

symbolic approach and some internal optimization techniques that PRISM uses to reduce the model size. For the same number of cryptographers, the CDC protocol model requires more states and transitions than the DC protocol model because of the increasing uncertainty. Generally speaking, the building model time for CDC is also slightly more than the building model time for DC (see part a of Figure 4.7). The execution time is the total time of constructing the model and computing iteratively the set of states which are reachable from the initial states and the transition matrix. As the number of cryptographers increases, the gap between construction time and execution time increases as well. This means the more cryptographers, the longer time is needed to compute reachability (see part b. of Figure 4.7).

We use C$i$paid to stand for cryptographer i paid the bill and Npaid for NSA paid the bill. p is the cheating index for cryptographers. We set 3 cryptographers in the systems and assume that all the cryptographers have the same cheating index for simplification reasons. $p = 0$ means that cryptographers do not cheat and follow the rules completely. Some desired properties of the protocol are expressed in Table 4.4 and are self-explanatory.

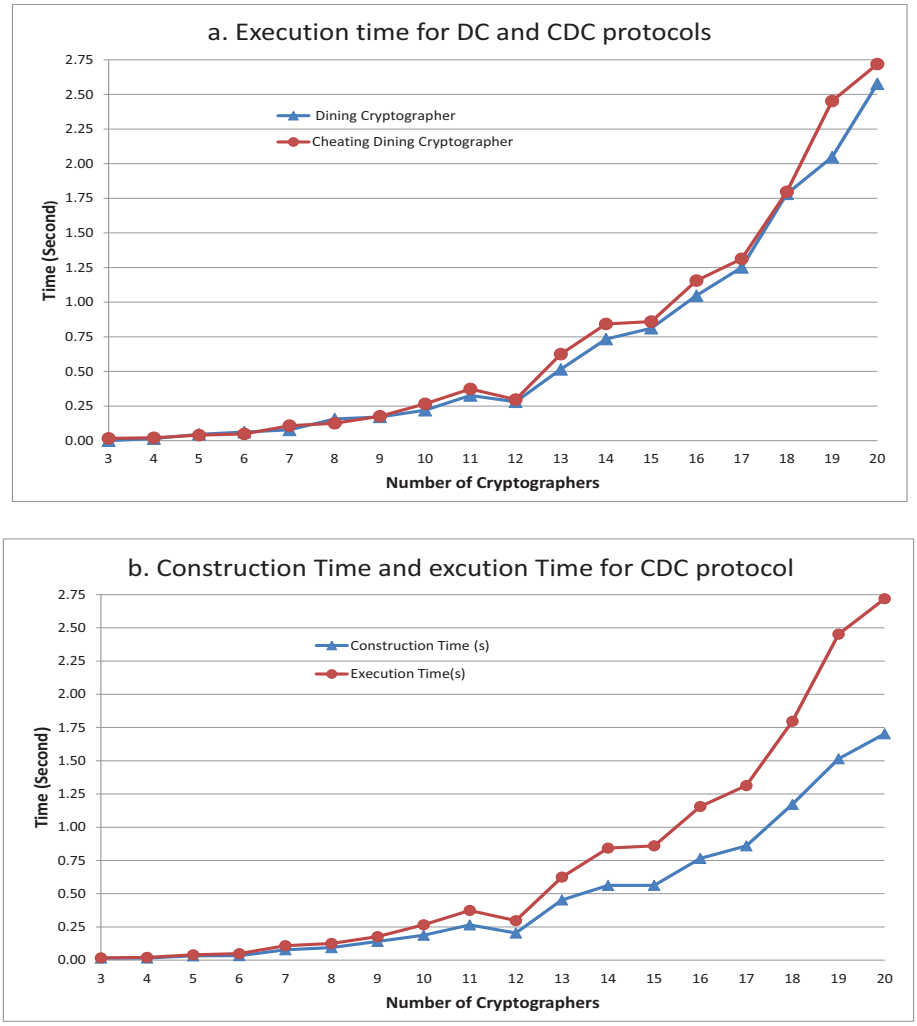

Figure 4.7: Construction and Execution Time for The CD and CDC Protocols

Table 4.4: Examples of Verified Properties for The CDC Protocol with 3 Cryptographers

No.	Formulae	Results	Time for Model Checking (sec.)
1	$(\text{even} \wedge p = 0) \Rightarrow \bigwedge_i (K_i(!\text{C}1\text{paid} \wedge !\text{C}2\text{paid} \wedge !\text{C}3\text{paid}))$	True	0.002
2	$(\text{odd} \wedge !\text{C}i\text{paid} \wedge p = 0) \Rightarrow (K_i(\bigvee_{j \neq i} \text{C}j\text{paid}))$	True	0.002
3	$(\text{odd} \wedge !\text{C}i\text{paid} \wedge p = 0) \Rightarrow K_i(P_{\leq 0.5}(\text{C}j\text{paid}))\quad j \neq i$	True (8/16=0.5)	0.005
4	$(\text{odd} \wedge !\text{C}i\text{paid} \wedge p = 0) \Rightarrow K_i(\text{C}j\text{paid}))\quad j \neq i$	False	0.002
5a.	(P = ? Npaid) (for $0 < p < 1$ step 0.1)	Figure 4.8: NSA pays	0.004
5b.	(P = ? $!N$paid) (for $0 < p < 1$ step 0.1)	Figure 4.8: Crypt pays	0.003
6a.	(P = ? even) (for $0 < p < 1$ step 0.1)	Figure 4.8: even difference	0.002
6b.	(P = ? odd) (for $0 < p < 1$ step 0.1)	Figure 4.8: odd difference	0.002
7a.	(P = ? (odd $\wedge !Npaid$)) (for $0 < p < 1$ step 0.1)	Figure 4.8: odd and Crypt paid	0.003
7b.	(P = ? (even $\wedge !Npaid$)) (for $0 < p < 1$ step 0.1)	Figure 4.8: even and Crypt paid	0.001
8a.	(P = ? (even $\wedge Npaid$)) (for $0 < p < 1$ step 0.1)	Figure 4.8: even and NSA paid	0.001
8b.	(P = ? (odd $\wedge Npaid$)) (for $0 < p < 1$ step 0.1)	Figure 4.8: odd and NSA paid	0.003

The properties expressed in the first four formulae are checked for the classical DC protocol since CDC and DC are equivalent when $p = 0$. The property encoded in formula 1 expresses that if NSA pay for the dinner and all the cryptographers are honest, an even number of differences is announced so that all the cryptographers know that the bill is paid by NSA. The properties expressed in the formulae 2 to 4 express that if cryptographer i does not pay the bill and an odd number of differences is stated, then i knows that the dinner is paid by one of his partners rather than NSA, but he does not know exactly who paid.

In PRISM, if ϕ is a formula, operator $P =?\phi$ is used to calculate the probability of ϕ. Formulae 5 to 8 show that the probability trends of certain properties vary as the cheating index changes from 0 (0% cheating) to 1 (100% cheating). The results of formulae 5a and 5b show respectively the probability that NSA and one of the cryptographers is paying the bill. The value does not change with cheating index because the probability of paying the bill is independent from this index. However, the number of differences announced changes with the cheating index (formulae 6a and 6b). When the cheating index increases from 0 to 1 the results become unpredictable because of the high uncertainty (Formulae 7a, 7b, 8a, and 8b).

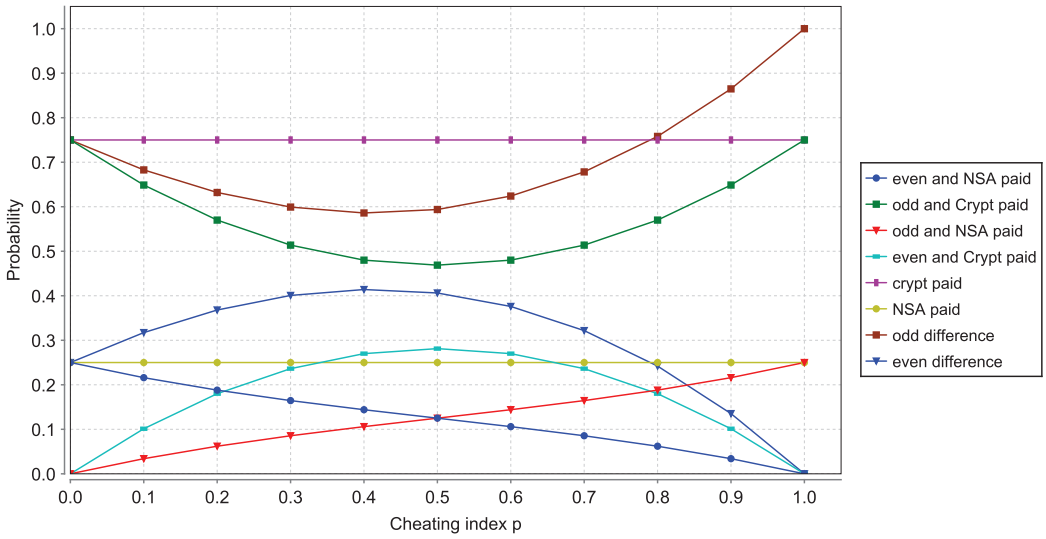

Figure 4.8: Verification Results of Some Properties in The CDC Protocol for 3 Cryptographers with Regard to The Cheating Index

4.4.4 Comparison with Existing Work

Although the DC protocol has already been modeled by many researchers [56, 60, 73, 94] in different ways to represent and reason about knowledge, most of them [60, 73, 94] only focus on qualitative properties of knowledge, so quantitative properties cannot be expressed in these approaches. Huang et al. in [56] present a symbolic model checking algorithm for the verification of probabilistic knowledge. Huang et al.'s approach works on a particular class of interpreted systems called partially observed discrete-time Markov chains (PO-DTMC), which are synchronous with perfect recall. The model checking algorithm has been implemented on top of the MCK model checker. Here we will compare the performance of three model checkers: MCK, MCMAS, and PRISM for verifying epistemic properties and probabilistic properties.

We verified two properties of the DC protocol, one epistemic and one probabilistic, using Huang et al.'s method with extended MCK and our approach with PRISM. MCK is a model checker for the logic of knowledge developed in the School

of Computer Science and Engineering at the University of New South Wales. It supports both linear and branching temporal logics with several different ways of defining knowledge based on the observations made by the agents: observation alone, observation and clock, and perfect recall of all observations. MCMAS is a symbolic model checker that is designed for MAS. We verify the epistemic property with MCMAS in this case study because MCMAS cannot check probabilistic properties. The comparison is under Fedora 16 with 3.1.0 Linux on the same computer for the experimental results (1.86 GHz Intel Core Duo T6300 processor and 3.25 GB memory).

The first property is that if the first cryptographer did not pay for the dinner, he knew that either NSA paid or one of the cryptographers paid, but he did not know which particular cryptographer paid. This epistemic property for three cryptographers can be expressed in MCK as follows:

$$spec\_spr\_xn = Xn(neg\ paid[0]) \Rightarrow ((Knows\ C1\ ((neg\ paid[1]) \wedge (neg\ paid[2])))$$
$$\vee((Knows\ C1\ (paid[1] \vee paid[2]) \wedge neg\ Knows\ C1\ paid[1]) \wedge neg\ Knows\ C1\ paid[2])) \tag{4.13}$$

where *spec_spr_xn* is a specification identifier indicating that agents's knowledge is based on synchronous perfect recall and the verification uses binary decision diagram symbolic model checking algorithm. *neg* is a negation operator and *Knows* is a knowledge operator. *paid[i]* is a Boolean variable that indicates if cryptographer i paid the bill. n is an integer stating that in n steps the specification holds and X is the next operator. The same property is expressed in PRISM (Formula (5.3)) and MCMAS (formula (4.15)) respectively as follows:

$$filter(forall, !\text{“}C0paid\text{”} \Rightarrow (P >= 1\ [X!\text{“}C1paid\text{”}]\&\ P >= 1[X!\text{“}C2paid\text{”}])$$
$$|((P >= 1[X!\text{“}C1paid\text{”}]|\ P >= 1[X!\text{“}C2paid\text{”}])$$
$$\&\ !P >= 1[X\text{“}C1paid\text{”}]\&\ !P >= 1[X\text{“}C2paid\text{”}]), \text{“}done\text{”}\&\ !\text{“}C0paid\text{”}) \quad (4.14)$$

$$AG((odd \text{ and } !c1paid) \rightarrow (K(C1, \ c2paid \text{ or } c3paid))$$
$$\text{and } !K(C1, \ c2paid) \text{ and } !K(C1, \ c3paid)); \quad (4.15)$$

Formula (5.3), where *filter* returns values for the identified set of states which satisfy the filter command. In this case, the system will compute values for all states where cryptographer $C1$ did not pay the bill.In formula (4.15), c1paid, c2paid, c3paid are defined in Evaluation section to represent the fact that Cryptographer 1, 2, and 3 paid the bill.

The second property is probabilistic. It indicates that one cryptographer knows that the other cryptographers have the equal probability to pay the bill. The probability is either 0 or $1/(n-1)$ for n cryptographers. In MCK, the probabilistic property is expressed as follows:

$$spec\_spr\_xn = Xn(negpaid[0]) \Rightarrow ((Prob\ C1\ paid[1] == Prob\ C2\ paid[2]) \wedge$$
$$((Prob\ C1\ paid[1] == 0) \vee (Prob\ C1\ paid[1] == 0.5))) \quad (4.16)$$

Although it is easy to calculate path probability in PRISM, it cannot compute the probability of knowledge directly. However, when we verify a filter specification, PRISM gives the number of states satisfying the filter (n_1) and the total number of

filtered states (n_2). Based on our definition of probabilistic knowledge, we are able to calculate the knowledge probability by computing n_1/n_2. So, in PRISM, we use the calculated result of the following expression to get the probability of knowledge:

$$filter(forall,\ P >= 1[X\text{“}C2paid\text{”}], \text{“}done\text{”}\&!\text{“}C1paid\text{”}\&\text{“}odd\text{”}) \tag{4.17}$$

Table 4.5: Experimental Results with MCK, PRISM and MCMAS

Number of Crypt. (n)	Time for MCK MC		Time for Extended PRISM MC		Time for MCMAS
	Epistemic property (sec)	Probabilistic property (sec)	Epistemic property (sec)	Probabilistic property (sec)	Epistemic property (sec)
3	0.071	1.405	0.003	0.006	0.011
4	0.382	136.6	0.004	0.018	0.014
5	6.181	-	0.005	0.053	0.055
6	-	-	0.011	0.063	0.116
7	-	-	0.019	0.138	0.21
8	-	-	0.022	0.254	0.534
9	-	-	0.033	0.48	0.943
10	-	-	0.044	0.863	1.844
11	-	-	0.059	1.445	2.219
12	-	-	0.079	2.366	9.883
13	-	-	0.109	3.741	-
14	-	-	0.124	5.756	-
15	-	-	0.216	8.476	-
16	-	-	0.224	12.301	-
17	-	-	0.287	17.695	-
18	-	-	0.352	24.567	-
19	-	-	0.403	33.546	-
20	-	-	0.607	45.686	-

Table 4.5 reports the runtime for verifying the epistemic and probabilistic properties with extended MCK, PRISM and MCMAS. With MCK, we can only the get runtime of verifying (1) the epistemic property for up to 5 cryptographers and (2) the probabilistic property for up to 4 cryptographers. With MCMAS, the case can be scaled up to 12 agents. However, with PRISM, we can get the verification results for up to 20 cryptographers. The runtime for verifying the DC protocol including 20 cryptographers with PRISM is still relatively low, which shows that our approach

is time efficient and scalable. In fact, for all the numbers of cryptographers ranging from 3 to 20, PRISM shows better performance for both epistemic and probabilistic properties. The results indicate that for both model checkers, namely extended MCK and PRISM, probabilistic properties need longer time to be verified, but our reduction-based model checking technique outperforms Huang et al.'s approach (See Figure 4.9). Also, as the size of the system increases, the execution time of all three model checkers increases: with MCMAS and PRISM, the increase is polynomial with a lower rate for PRISM, whereas with extended MCK increases is exponential.

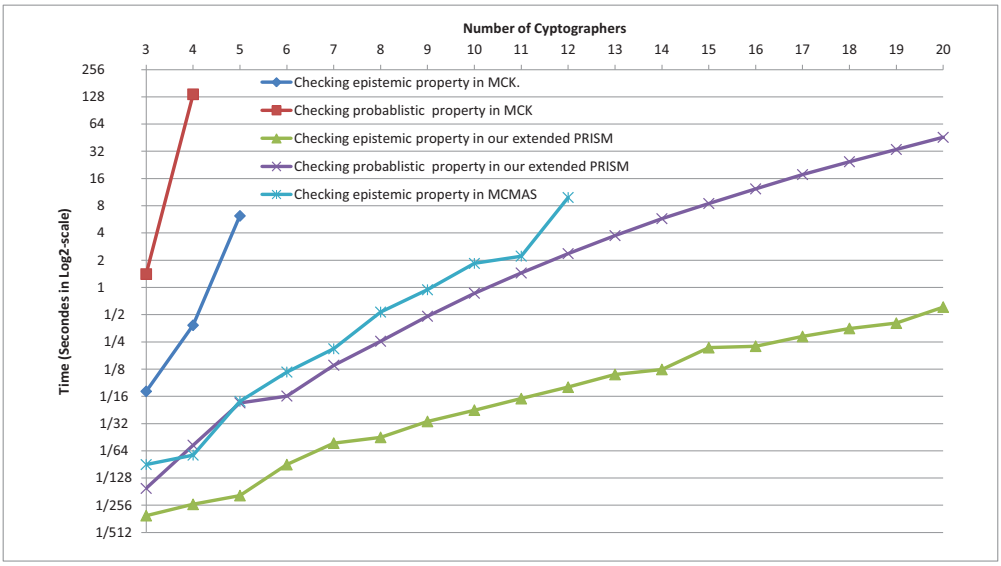

Figure 4.9: Runtime for Verifying The DC Protocol

The experimental results show that in verifying the DC protocol, PRISM has the best performance. This is because in the DC protocol, properties are checked after all the cryptographers have finished their actions. The verification only happens on the static environment. In fact, PRISM is designed for quantitative verification, while

MCK and MCMAS are for knowledge verification. Our method reduces knowledge verification into probabilistic computation tree logic verification, so it benefits from the efficiency of PRISM in verifying such a logic. thanks to the Multi-terminal Binary Decision Diagram (MTBDD) package used in this model checker,compared to the limited Binary Decision Diagram (BDD) package used in MCK and MCMAS. However for reactive dynamic systems that need to recall previous settings, PRISM does not function well. In contrast, MCK is good at verifying such systems using observations with perfect synchronous recall. MCK is designed for supporting several different ways of defining knowledge, including synchronous and asynchronous perfect recall for observations. MCMAS is tailored to the verification of MAS scalable systems. It can be used to verify group knowledge and collaboration among agents.

Chapter 5

Model Checking Knowledge in Concurrent Probabilistic Systems

This chapter is largely based on our manuscript submitted to *Journal of Computers and Mathematics with Applications* [98]. Our research question 5 is answered in this chapter. In Section 5.1, we review the definitions of related concepts, such as concurrent probabilistic systems and adversaries. In Section 5.2, we explain how model checking PCTLK can be reduced to model checking PBTL. This reduction process includes two steps:

1. Model reduction: a PCTLK probabilistic interpreted system is reduced to PBTL structure (Section 5.2.1), and

2. Formulae transformation: serial rules for converting a PCTLK formula to a PBTL formula (Section 5.2.2).

The complexity of PCTLK model checking over concurrent probabilistic programs is analyzed in Section 5.3. and space complexity (Section 5.3.2) are proved to be polynomial functions in the size of the model and length of the formula. Section 5.4

discusses symbolic representation of probabilistic interpreted systems with MTBDDs data structure. We implement our approach with PRISM [69] in Section 5.5 and analyze the experimental results.

5.1 Concurrent Probabilistic Systems

Concurrency is a property of MAS and other complicated distributed systems in which several computations are executed simultaneously and potentially interact with each other. When we use model checking to verify practical systems, they are usually composed of several explicit models in concurrent systems [12, 65] rather than only one single explicit model. We consider concurrent probabilistic systems in this chapter, in which several probabilistic systems work asynchronously and nondeterministically.

Concurrent probabilistic systems exhibit probabilistic and nondeterministic choices. They can be generated by Markov decision processes [7, 97]. The probabilistic choices are made by the system itself. Probabilistic systems can be formally expressed using Markov chains, such as Discrete-Time Markov Chains (DTMCs) or Continuous-Time Markov Chains (CTMCs). Because our logic is discrete time-based, we use DTMCs to express the probabilistic choices.

The nondeterministic choices are caused by several components in the systems working asynchronously. These kinds of choices are beyond the control of the process. Concurrent probabilistic systems can be seen as combinations of several interactive DTMC models. Such systems allow several probability distributions to be enabled in a given state. Baier and Kwiatkowska defined concurrent probabilistic systems in [7] as a pair $\mathscr{P} = (S, Steps)$.

Definition 5.1. (Concurrent Probabilistic Systems)

A concurrent probabilistic system can be expressed as $\mathscr{P} = (S, Steps)$*, where*

- *S is a finite set of states.*
- *Steps is a function that assigns to each state $s \in S$ a finite, non-empty set of probability distributions μ_i; each μ_i is a vector of probabilities, $\mu_i = (v_0, ..., v_n)$, such that $\sum_{x=0}^{n} v_x = 1$.*

We use the notation $\mu_i[x]$ to indicate the value of the distribution μ_i at the index x. *Steps* represents the nondeterministic alternatives in each state. Every element in *Steps* represents one probability distribution in the system. For example, for states $s, t \in S$, one possible distribution μ_i from $Steps(s)$ will be associated with a DTMC model, such that $\mu_i[x] = \mathbf{P}(s,t)$ for some index x in the vector μ_i. When the target state t is known, we use the notation $\mu_i(t)$ instead of $\mu_i[x]$.

Let $s_i \in S$ be a state, $\mu_i \in Steps(s_{i-1})$ and $\mu_i(s_i) > 0$, a path in a concurrent probabilistic system is a sequence of states related by determinate transitions: $\pi = s_0 \xrightarrow{\mu_1(s_1)} s_1 \xrightarrow{\mu_2(s_2)} s_2 \ldots$. The $(i+1)th$ state in a path π is denoted $\pi(i)$ i.e., $\pi(i) = s_i$. If π is a finite path, we denote $last(\pi)$ as the last state of π and $|\pi|$ as the length of the path. We also use σ to stand for a finite path. and $\sigma(s)$ to stand for all finite paths emanating from s.

The nondeterminism of a concurrent probabilistic system can be determined with *an adversary* (alternatively called *a scheduler*). An *adversary* is a function from state set to *Steps* set. By selecting the same distribution every time when a given state is reached, an *adversary* converts the nondeterministic choices into probabilistic choices. In [70], Kwiatkowska et al. defined an *adversary* as follows:

Definition 5.2. (Adversary of a Concurrent Probabilistic System)
An adversary (or scheduler) of a concurrent probabilistic system $\mathscr{P} = (S, Steps)$ is a function $\mathscr{A}$ mapping every state s of S to a a distribution on S.

From this definition, we can see that an *adversary* $\mathscr{A}$ of the concurrent probabilistic system $\mathscr{P}$ chooses the specific *Steps* to turn nondeterministic choices into deterministic ones, which can be expressed as a DTMC model with a transition probability function $\mathbf{P}$ as shown in Equation (5.1). $Path^{\mathcal{A}}(s)$ is used to express the set of all path with the adversary $\mathcal{A}$ starting from state s. (we use $\mathbf{P}^{\mathscr{A}}$ to represent the transition probability function of the restricted concurrent probabilistic system by the adversary $\mathscr{A}$, and μ instead of μ_i when there is no confusion):

$$\mathbf{P}^{\mathscr{A}}(s,s') = \begin{cases} \mu(s') = \mathbf{P}(s,s'), & \text{if } \mathscr{A}(s) = \mu \text{ and } s \xrightarrow{\mu(s')} s' \\ 0, & \text{otherwise} \end{cases} \tag{5.1}$$

5.2 Model Checking PCTLK in Concurrent Systems

A model checking approach based on reduction for PCTLK specification has been discussed in Chapter 4 and proposed in [102] which transforms the problem of model checking PCTLK into the problem of model checking PCTL. In this chapter, we will transform our model into a PBTL structure that includes a concurrent probabilistic system and transform a PCTLK formula into a PBTL formula. Given a PCTLK model M_{PIS} and a PCTLK formula ϕ_{PIS}, we can define a concurrent probabilistic system for PBTL structure $M = \mathcal{F}(M_{PIS})$ and a PBTL formula $\varphi = \mathcal{F}(\varphi_{PIS})$ such that $(M_{PIS}, s_0) \models \varphi_{PIS}$ iff $(\mathcal{F}(M_{PIS}), s_0) \models \mathcal{F}(\varphi_{PIS})$. A PCTLK model M_{PIS} can be considered as a system that incorporates or independently joins n PBTL models with S5^n [16, 58], where n is the number of agents in the system.

In the following, we split the explanation into two parts: models reduction and formulae reduction to introduce the details of model checking concurrent probabilistic

systems against PCTLK specification.

5.2.1 Models Reduction

We first analyze the similarities and differences between a probabilistic interpreted system $M_{PIS} = (W, \mathbf{P_t}, I_{init}, \sim_i, AP, V)$ $(0 < i \leq n)$ and a PBTL structure $C = (\mathscr{P}, I, AP, L)$. Both models have the same atomic proposition set (AP), the same labeling functions (V and L), and the same initial distribution I_{init} and I. We can set these values directly. The differences are:

1. There are a set of state W, probabilistic transition function $\mathbf{P_t}$, and epistemic accessibility relations $\sim_i$ in an M_{PIS} model but not in a PBTL structure C.

2. A pair $\mathscr{P}$ is in C structure while it does not appear in M_{PIS}.

However there are hidden links between these two differences. A pair $\mathscr{P} = (S, Steps)$ in a PBTL structure includes a set of states S and a set of functions $Steps$ whose elements are probabilistic transitions for every state. Moreover, if we consider epistemic accessibility relations $\sim_i$ as agent i's special optional transition $\boldsymbol{Ag}_i$ on the concurrent probabilistic system based on its epistemic accessibility states, we are able to convert these relations into $Steps$ functions. We can see that a concurrent probabilistic system is obtained by the parallel composition of n agents' behavior. A state in a concurrent probabilistic system will be a tuple $(s_1, \ldots, s_n)$, where s_i $(i \in \{1, \ldots, n\})$ is a state in agent i's system. Therefore, model M_{PIS} is a special concurrent probabilistic system where two states $s = (s_1, \ldots, s_n)$ and $s' = (s'_1, \ldots, s'_n)$ are related for agent i's program if and only if $s_i = s'_i$.

Based on our analysis, we know that the key to transform a probabilistic interpreted model M_{PIS} into a PBTL structure C_{IS} is to find a method to map the epistemic accessibility relations $\sim_i$ to a specific step $\boldsymbol{Ag}_i$. Epistemic accessibility

relations for agent $\sim_i$ define agent i's possible world in global states. Agent i will see the same changes/unchanges in its possible world. For example, if $(s,t) \in \sim_i$ and $(s,w) \in \sim_i$, agent i cannot tell the difference among s,t,w at state s. We can see these relations as a special transition $\boldsymbol{Ag}_i$ indicates that global states change but agent i cannot be aware of these changes. The probability of these kind of transitions can be decided by how many global states there are in the agents' possible world.

Now we can define the PBTL structure $C_{IS} = (\mathscr{P}_{IS}, AP_{IS}, L_{IS}, I_{IS})$ obtained from our M_{PIS} model.

Definition 5.3. Structure C_{IS}

Given a probabilistic interpreted model $M_{PIS} = (W, \mathbf{P_t}, I_{init}, \sim_1, \ldots, \sim_n, AP, V)$, $C_{IS} = (\mathscr{P}_{IS}, I_{IS}, AP_{IS}, L_{IS})$ is a PBTL structure defined from M_{PIS} as follows:

- *$\mathscr{P}_{IS} = (S_{IS}, Steps_{IS})$ is a concurrent probabilistic system, where:*
  - *$S_{IS} = W$; and*
  - *$Steps_{IS}$ is a function which assigns to each state $s \in S_{IS}$ a finite, non-empty set $Steps(s)$ of distributions on S_{IS}. For every $s' \in S_{IS}$, the probability distribution $\mathbf{P_{IS}}^{\mathscr{A}}(s,s')$ is defined by Equation (5.2).*

$$\mathbf{P_{IS}}^{\mathscr{A}}(s,s') = \begin{cases} \mu(s') = \mathbf{P_t}(s,s'), & \text{if } \mathscr{A}(s) = \mu \text{ and } s \xrightarrow{\mu} s' \\ \frac{1}{|s \sim_i s'|}, & \text{if } \mathscr{A}(s) = \boldsymbol{Ag}_i \text{ and } s \xrightarrow{\boldsymbol{Ag}_i} s' \\ 0, & \text{otherwise} \end{cases} \tag{5.2}$$

- *$I_{IS} = I_{init}$;*
- *$AP_{IS} = AP$; and*
- *$L_{IS} = V$.*

According to this transformation, the PBTL C_{IS} structure has the same initial distribution and global state labeling function over the same given set of atomic propositions as the M_{PIS} model. The labels of the transition set of C_{IS} include $\{\mu, Ag_1, \ldots, Ag_n\}$, where μ represents the label of the regular transitions of M_{PIS}, and $\boldsymbol{Ag}_i$ represents the epistemic accessibility relation of agent i. After this mapping, transitions and epistemic accessibility relations are associated with the *Steps* in a concurrent probabilistic system. The probabilities on the regular transitions μ of C_{IS} are obtained from the transitions of M_{PIS} with the same probabilities. For simplicity, we assume that epistemic accessibility relations are uniformly distributed among agents' possible world states. Therefore, the probability of transitions on adversary $\boldsymbol{Ag}_i$ for a state can be obtained by the inverse of the total number of accessible states. For example, if 3 states are accessible from a state s for agent i, then each of the the three obtained transitions will have 1/3 probability for the label $\boldsymbol{Ag}_i$.

Because C_{IS} is a nondeterministic system, a policy or strategy should be used to resolve all the nondeterministic choices. We use an adversary, which was introduced in Definition 5.2 to make the concurrent probabilistic systems deterministic. We divide adversaries into two types: one is a regular probabilistic transition adversary that we represent as λ and another the transition converted from an agent epistemic accessibility relation that we denote as λ_i. Here we show how two particular adversaries λ and λ_i are used to obtain a deterministic system from a concurrent probabilistic system for PBTL structure C_{IS}.

From each state in C_{IS} there are two kinds of enabled steps: regular transition μ and epistemic relation $\boldsymbol{Ag}_i$. We set the adversary λ that always selects the transitions labeled by regular transition μ (so the transitions obtained by the accessibility relations are ignored). The adversary λ_i always selects the steps $\boldsymbol{Ag}_i$ from a state

s and then selects the regular transiton μ from all the following states (so first the accessibility relations $\sim_i$ are considered and then the normal transitions).

Next we will introduce the set of reduction rules for PCTLK formulae.

5.2.2 Formula Reduction

In this section, we will explain how to reduce PCTLK formulae into PBTL formulae. Most PBTL and PCTLK formulae have the same operators and syntax except:

1. PBTL allows universal ($\forall$) and existential ($\exists$) path quantifiers while path quantifiers are not eligible in a PCTLK formula; and

2. PCTLK includes the epistemic formula $K_i\phi$ and probability of epistemic formula $Pr_{\nabla b}K_i\phi$ but PBTL does not.

Based on the syntax of PCTLK, we can conclude that all PCTLK path formulae are equivalent to PBTL path formulae with existential quantifiers. Therefore, if we add the existential quantifier $\exists$ to a PCTLK path formula, the PCTLK path formula will be a PBTL path formula. From the semantics of the epistemic formula $K_i\phi$, we can imagine that if formula ϕ is satisfied in all agent i's epistemic accessibility states, then $K_i\phi$ holds. Thus, we can transform the epistemic operator K_i to the next operator based on the epistemic accessibility relation $\sim_i$.

This reduction process, similar to the process presented in the previous chapter, works in stages inductively. We first divide each PCTLK formula into "maximal state subformulae" such that each maximal state subformula ϕ: (1) includes the probabilistic $Pr_{\nabla b}$ or epistemic operator K_i; (2) differs from ϕ; and (3) is not contained in any other state subformula of ϕ. Each maximal state subformula is a PCTLK formula, which can be divided into other maximal state subformulae until no new maximal state subformula can be identified. Thus, in stage k, formulae of level k

are decomposed into maximal state subformulae of level $k-1$ or lower. The state subformulae of level k are defined inductively as follows:

- Level 0 contains atomic propositions;
- Level $(k+1)$ $(k \geq 0)$ contains all stage subformulae ϕ' such that all state subformulae of ϕ' are of level k or less and ϕ' is not contained in any lower level.

Therefore, the lowest level, level 0, contains only atomic propositions and the highest level is the whole formula. In stage k all maximal state subformulae of ϕ of level smaller than k are processed and replaced by new atomic propositions, which are labels of the subformulae. Before giving an example, we introduce the reduction rules. Let $\phi,\ \phi_1,\ \phi_2$ be PCTLK formulae, a be an atomic proposition and $\mathcal{F}(\phi)$ be a PBTL formula that transforms from PCTLK; the transformation rules are defined recursively as in Table 5.1.

Table 5.1: PCTLK to PBTL Transformation Rules

Rule Number	PCTLK subformula	converted subformula
1	a	$\mathcal{F}(a)$
2	$\neg\phi$	$\neg\mathcal{F}(\phi)$
3	$\phi_1 \wedge \phi_2$	$\mathcal{F}(\phi_1) \wedge \mathcal{F}(\phi_2)$
4	$Pr_{\triangledown b} \bigcirc \phi$	$Pr_{\triangledown b}\exists \bigcirc \mathcal{F}(\phi)$
5	$Pr_{\triangledown b}(\phi_1 \mathrm{U} \phi_2)$	$Pr_{\triangledown b}(\mathcal{F}(\phi_1)\exists \mathrm{U} \mathcal{F}(\phi_2))$
6	$Pr_{\triangledown b}(\phi_1 \mathrm{U}^{\leq n} \phi_2)$	$Pr_{\triangledown b}(\mathcal{F}(\phi_1)\exists \mathrm{U}^{\leq n} \mathcal{F}(\phi_2))$
7	$K_i\phi$	$Pr_{\geq 1}(\exists \bigcirc \mathcal{F}(\phi))$
8	$Pr_{\triangledown b}K_i\phi$	$Pr_{\triangledown b}(\exists \bigcirc \mathcal{F}(\phi))$

To complete the reduction process, a step for the reduced concurrent probabilistic model $\mathcal{F}(M_{PIS})$ associated with each PBTL formula should be defined. This is done by specifying which adversary is associated with which formula. In the following, $(\mathcal{F}(M_{PIS}), s) \models_\lambda \phi$ means the PBTL formula ϕ is satisfied in the model $\mathcal{F}(M_{PIS})$ obtained by applying the adversary regular λ at state s, while $(\mathcal{F}(M_{PIS}), s) \models_{\lambda_i} \phi$ is associated with agent i's epistemic accessibility relation at state s. The following

theorem is a direct consequence of the definition of $\mathcal{F}$ and can be easily proved by induction on the structure of the formula.

Theorem 5.1. Satisfaction Equivalence

$$(M_{PIS}, s) \models a \ \textit{iff} \ (\mathcal{F}(M_{PIS}), s) \models_{\lambda} a$$
$$(M_{PIS}, s) \models \neg\phi \ \textit{iff} \ (\mathcal{F}(M_{PIS}), s) \models_{\lambda} \neg\mathcal{F}(\phi)$$
$$(M_{PIS}, s) \models \phi_1 \wedge \phi_2 \ \textit{iff} \ (\mathcal{F}(M_{PIS}), s) \models_{\lambda} \mathcal{F}(\phi_1) \wedge \mathcal{F}(\phi_2)$$
$$(M_{PIS}, s) \models Pr_{\triangledown b} \bigcirc \phi \ \textit{iff} \ (\mathcal{F}(M_{PIS}), s) \models_{\lambda} Pr_{\triangledown b} \exists \bigcirc \mathcal{F}(\phi)$$
$$(M_{PIS}, s) \models Pr_{\triangledown b}(\phi_1 \mathrm{U} \phi_2) \ \textit{iff} \ (\mathcal{F}(M_{PIS}), s) \models_{\lambda} Pr_{\triangledown b}(\mathcal{F}(\phi_1) \exists \mathrm{U} \mathcal{F}(\phi_2))$$
$$(M_{PIS}, s) \models Pr_{\triangledown b}(\phi_1 \mathrm{U}^{\leq n} \phi_2) \ \textit{iff} \ (\mathcal{F}(M_{PIS}), s) \models_{\lambda} Pr_{\triangledown b}(\mathcal{F}(\phi_1) \exists \mathrm{U}^{\leq n} \mathcal{F}(\phi_2))$$
$$(M_{PIS}, s) \models K_i\phi \ \textit{iff} \ (\mathcal{F}(M_{PIS}), s) \models_{\lambda_i} Pr_{\geq 1}(\exists \bigcirc \mathcal{F}(\phi))$$
$$(M_{PIS}, s) \models Pr_{\triangledown b}(K_i\phi) \ \textit{iff} \ (\mathcal{F}(M_{PIS}), s) \models_{\lambda_i} Pr_{\triangledown b}(\exists \bigcirc \mathcal{F}(\phi))$$

Simply put, this theorem states that if the PCTLK formula does not include epistemic operator K_i, then the corresponding PBTL formula is satisfied a *Steps* distribution in the concurrent probabilistic systems obtained by only considering the normal transitions. However, if the formula has the form of $K_i\phi$, then the corresponding PBTL formula is satisfied in the *Steps* obtained by considering first the transition associated with the epistemic accessibility relation $\sim_i$ and then the normal transitions, which shows why the K operator is transferred to the next operator.

In fact, PCTLK language can be seen as a variant of CTLK [85] by replacing the path quantifiers $\exists$ and $\forall$ by the probabilistic operator Pr. PCTLK provides possibility b for path formulae to specify the likelihood of the path. The critical values, 0 and 1, state two specific situations: $Pr_{\geq 1}(\psi)$ stands for the path formula ψ happens all the time; while $Pr_{\leq 0}(\psi)$ represents the negation of the path formula ψ. Therefore, when we constrain that the probabilistic relation can only be "≥ 1" or "> 0", PCTLK turns into CTLK: $Pr_{\geq 1}$ is equivalent to $\forall$ and $\mathrm{Pr}_{>0}$ is equivalent to $\exists$.

PCTLK also is an extension of PCTL [6, 52] as it includes the epistemic operator K_i. The properties expressed using PCTL can be checked by the PRISM model checker [67]. Therefore, based on PCTLK syntax, a formula can be a CTL formula, a CTLK but not PCTL formula, a PCTL but not CTLK formula, a conjunction of CTLK and PCTL like $K_i\phi \wedge Pr_{>0.5}(\psi)$, or a pure epistemic probabilistic formula under the form $K_i(Pr_{\triangledown b}(\psi))$. The structure of $PCTLK$ is shown in Fig. 5.1.

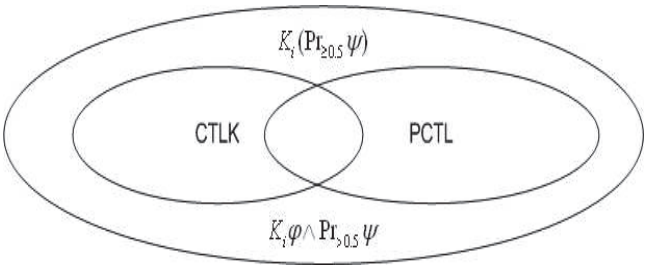

Figure 5.1: Structure of PCTLK

Theorem 5.2. Soundness and Completeness

Let M_{PIS} a probabilistic interpreted system and ϕ be a PCTLK formula , and let $\mathcal{F}(M_{PIS})$ and $\mathcal{F}(\phi)$ be the corresponding PBTL structure and PBTL formula. Then $M_{PIS} \models \phi$ if and only if $\mathcal{F}(M_{PIS}) \models_\lambda \mathcal{F}(\phi)$ or $\mathcal{F}(M_{PIS}) \models_{\lambda_i} \mathcal{F}(\phi)$, based on the formula being without or with the epistemic operator K_i.

Proof.

We prove this theorem by induction on the structure of the formula ϕ. The rules 1 to 3 are straightforward (see Table 5.1). We need to consider the rules 4 to 8.

- Rule 4 for formula $\phi = Pr_{\triangledown b} \bigcirc \psi$. We have that $(M_{PIS}, s) \models Pr_{\triangledown b} \bigcirc \psi$ if and only if $Prob(s, \sigma(s), \bigcirc\psi)\ \triangledown b$, where $Prob(s, \sigma(s), \bigcirc\psi) = \sum_{\substack{\pi \in \sigma(s) s.t. \\ \pi(1) \models \psi}} \mathbf{P_t}(s, \pi(1))$. For adversary $\mathscr{A}(s) = \lambda$, we know that $\mathbf{P_{IS}}^{\mathscr{A}}(s, \pi(1)) = \mathbf{P_t}(s, \pi(1))$. Therefore, $Prob(s, \sigma(s), \bigcirc\psi) = Prob(\pi \in Path^\lambda(s) : (\mathcal{F}(M_{PIS}, \pi) \models_\lambda \bigcirc\mathcal{F}(\psi))$. According to $Pr_{\triangledown b}\exists\bigcirc$ semantics for PBTL, we obtain $(\mathcal{F}(M_{PIS}), s) \models_\lambda Pr_{\triangledown b}\exists\bigcirc\mathcal{F}(\psi)$.

On the other hand, for adversary λ, if $(\mathcal{F}(M)_{PIS}, s) \models_{\lambda} \mathrm{Pr}_{\triangledown b} \exists \bigcirc \mathcal{F}(\psi)$, it is straightforward that $(M_{PIS}, s) \models Pr_{\triangledown b} \bigcirc \psi$. The same method can be used to prove Rule 5 and Rule 6.

- Rule 7 for formula $\phi = K_i\psi$. We have that $(M_{PIS}, s) \models K_i\psi$ iff $\forall s' \in W$, if $s \sim_i s'$, then $(M_{PIS}, s') \models \psi$. Consequently, $\forall s' \in S_{IS}$, if $\mathscr{A}(s) = Ag_i$, then it follows that every infinite path π emanating from s satisfies that $\pi(1) = s'$ and $(\mathcal{F}(M_{PIS}), s') \models_{\lambda_i} \mathcal{F}(\psi)$. By the semantics of $Pr_{\geq 1}\exists\bigcirc$, and $\frac{\sum_1^{|s\sim_i s'|} 1}{|s\sim_i s'|} = 1$, we obtain $(\mathcal{F}(M_{PIS}), s) \models_{\lambda_i} Pr_{\geq 1}(\exists \bigcirc \mathcal{F}(\psi))$, and vice versa.

- Rule 8 for formula $\phi = Pr_{\triangledown b}K_i\psi$. We have that $(M_{PIS}, s) \models Pr_{\triangledown b}K_i\psi$ iff $Prob(s \models K_i\psi) \triangledown b$, where $Prob(s \models K_i\psi) = \frac{\sum_{s\sim_i s'} |s' \models \psi|}{|s\sim_i s'|}$. Recall that the probability transitions $\mathbf{P_{IS}}^{\mathscr{A}} = \frac{1}{|s\sim_i s'|}$ for $\mathscr{A}(s) = Ag_i$ at state s; therefore, for adversary $\mathscr{A}(s) = Ag_i$, $Prob(\pi \in Path^{\lambda_i}(s) : (\mathcal{F}(M_{PIS}, s') \models_{\lambda_i} \mathcal{F}(\psi))$ $= \sum_{\substack{\pi\in\sigma(s) s.t. \\ \pi(1)\models_{\lambda_i}\mathcal{F}(\psi)}} \mathbf{P_{IS}}^{\mathscr{A}}(s, \pi(1)) = \frac{\sum_{s\sim_i s'} |s' \models_{\lambda_i} \mathcal{F}(\psi)|}{|s\sim_i s'|}$, where $\pi(1) = s'$. By the semantics of $\exists\bigcirc$, we obtain $(\mathcal{F}(M_{PIS}), s) \models_{\lambda_i} Pr_{\triangledown b}(\exists \bigcirc \mathcal{F}(\psi))$, and vice versa.

Therefore, the theorem. □

5.3 Complexity Analysis

Many researchers have already investigated the complexity of model checking concurrent systems against temporal logic [41, 65, 77]. The complexity of concurrent probabilistic systems against probabilistic temporal logic like PCTL specifications are also explored in [13, 24]. In [7], Baier and Kwiatkowska already proved that the time complexity of model checking for PBTL over concurrent probabilistic systems is a polynomial function of the size of the PBTL structure and a linear function of the size of the PBTL formula. Furthermore, these model checking problems can be solved

on space of $\mathcal{O}(|\phi| \cdot n + n \cdot m)$, where n is the number of states and m is the number of transitions in the PBTL structure and $|\phi|$ is the length of the formula.

In this section, we will analyze both the time complexity and space complexity of model checking PCTLK with respect to the size of the system model and the length of the formula.

5.3.1 Time Complexity

In this subsection, we will prove that the running time for model checking PCTLK for Markov chains is polynomial in the size of the explicit model and the length of the formula.

Theorem 5.3. *The time complexity of the model checking problem for PCTLK for Markov chains is polynomial in the size of the explicit model and the length of the formula.*

Proof. PCTLK extends PCTL and PCTL is a subset of PBTL [7]. Thus, PCTLK extends PBTL as well. It is known from [7] that the model checking problem for PBTL for Markov chains can be solved in time polynomial in the size of the explicit PBTL structure and linear in the length of the formula. Thus, we need only to analyze the time complexity of our model transformation and formula reduction.

To go from our model M_{PIS} to PBTL structure C_{IS}, atomic propositions AP_{IS}, set of states S_{IS}, initial distribution I_{IS}, and labeling functions L_{IS} can be obtained directly from M_{PIS}. Therefore, these parts of the transformation will run in a linear time in the size of the model. We need two stages to generate *Steps* for PBTL structure C_{IS}. First, we add the regular transition μ into *Steps*; the value of probabilistic transitions $\mathbf{P_{IS}}(s, s')$ can be copied from $\mathbf{P_t}(s, s')$ of our M_{PIS} model directly. This stage will run in $\mathcal{O}(k)$ time, where k is the number of transitions in the Markov chain.

Then, we convert each epistemic accessibility relation $\sim_i$ into a transition $\boldsymbol{Ag}_i$ and compute the value of this transition. To compute the probability of this transition, we need to count the number of epistemic accessible states for agent i at state s. This can run in $\mathcal{O}(m)$ time, where m is the number of states in the system. Thus, to convert $\mathbf{P}_{IS}$ we need $\mathcal{O}(n \cdot m)$ time, where n is the number of agents in the system. In other words, the complexity of generating *Steps* for the PBTL structure is quadratic in the size of the M_{PIS} model ($\mathcal{O}(k + n \cdot m)$). Therefore, the complexity of the whole transformation is quadratic in the size of the model structure.

To transform a PCTLK formula ϕ, the procedure will recursively apply the transformation rules until a PBTL $\mathcal{F}(\phi)$ subformula is encountered. Therefore, the depth of the recursion is bounded by the length of the formula ϕ. Thus, the transformation is linear in the length $|\phi|$.

Because model checking PBTL for Markov chains can be solved in polynomial time in the size of the explicit model and linear in the length of the formula [7], model transformation can be performed in quadratic time in the size of the model, and formula reduction can be executed in linear time in the length of the formula, we conclude that the complexity of model checking PCTLK for Markov chains is polynomial in the size of the explicit Markov chain and the length of the formula. □

Theorem 5.4. *The model checking problem for PCTLK over Markov chains is P-complete.*

Proof. Membership: From Theorem 5.3, we get that the upper bound of the model checking problem is polynomial in the size of the Markov chain and the size of the formula.

Hardness: to prove the hardness, we need to prove that every problem A in the class P is polynomial time reducible (we use $\leq_P$ to express polynomial time reduction) to the problem of model checking PCTLK for Markov chains. We know from [29,

88] that model checking CTL for explicit models is P-complete. Therefore, $A \leq_P MC(CTL)$ (we use $MC(X)$ to represent model checking problem of language X and $\mathcal{L}(X)$ to express the language of X). Also $\mathcal{L}(CTL) \leq_P \mathcal{L}((PCTL) \leq_P \mathcal{L}(PCTLK)$, (because all CTL formulae can be expressed in PCTL, and PCTL is a subset of PCTLK), and explicit systems can be seen as a subset of Markov chains with all probability transitions are 1. Consequently, problem A can be polynomially reduced to the model check problem of PCTLK for Markov chains ($A \leq_P MC(PCTLK)$)[1]. So we can conclude that the model checking problem for PCTLK for Markov chains is P-complete. □

5.3.2 Space Complexity

In this subsection, we will prove that the complexity of PCTLK model checking for concurrent probabilistic systems is PSPACE-complete.

Theorem 5.5. *Let $|\phi|$ be the size of the state formula ϕ, n the number of agents in the concurrent probabilistic system, m the number of states in the system, and k the number of transitions in the system. The model checking problem for PCTLK on this concurrent probabilistic system can be solved in space $\mathcal{O}(|\phi| \cdot m + m \cdot (k + n \cdot m))$.*

Proof. It is known from [7] that the space complexity for PBTL over a concurrent probabilistic system is $\mathcal{O}(|\phi'| \cdot m + m \cdot k)$, where $|\phi'|$ is the size of PBTL formula ϕ', m is the number of states, and k is the number of transitions in the system. In Section 5.2, we have presented transformations of probabilistic epistemic models M_{PIS} for PCTLK to concurrent probabilistic systems C_{IS} for PBTL and of a PCTLK formula ϕ to a PBTL formula $\mathcal{F}(\phi)$ so that $M_{PIS} \models \phi$ if and only if $C_{IS} \models \mathcal{F}(\phi)$. We just need to analyze if the space complexity of model transformation and formula transformation

[1] If a problem B is P-complete and $B \leq_P C$, where C is in the class P, then C is P-complete.

is polynomial. Next we prove that a deterministic Turing machine TM computes the model reduction in polynomial space in the size of the input PCTLK model.

TM reads in the input tape a model of PCTLK M_{PIS} and generates the PBTL structure C_{IS} in the output tape, one by one, the same atomic propositions, states, the initial distribution, and labeling functions as the input. Furthermore, TM writes $\mu(s')$ into $Steps_{IS}$ for state s when TM reads a probabilistic transition function $\mathbf{P_t}(s, s')$ from the input model. TM reads epistemic relations $s \sim_i s'$ from the input model one by one and for each one, it writes $\boldsymbol{Ag}_i$ into $Steps_{IS}$ for state s in the output tape. These two steps are executed in space $\mathcal{O}(k + n \cdot m)$, where k is the number of transitions, m is the number of states, and n is the number of agents in the system. In fact, the generated PBTL structure has m states and $k + m \cdot n$ transitions.

On the other hand, from Table 5.1, we can see that the length of the PBTL formula $\mathcal{F}(\phi)$ is linear in the length of the original formula ϕ.

Based on complexity of our reductions and the result from [7], we can conclude that the model checking problem for PCTLK on a concurrent probabilistic system can be solved in space $\mathcal{O}(|\phi| \cdot m + m \cdot (k + n \cdot m))$, where $|\phi|$ is the length of the PCTLK formula. □

Theorem 5.6. *The space complexity of PCTLK model checking for a concurrent probabilistic program is PSPACE-complete.*

Proof. Membership in PSPACE follows from Theorem 5.5.

To prove the hardness, we need to prove that every problem A in PSPACE can be polynomially reduced to the problem of model checking PCTLK over concurrent probabilistic systems. We know that model checking CTLK for concurrent systems is PSPACE-complete [77]. Therefore, $A \leq_P MC(CTLK)$. Moreover, the CTLK model checking problem over concurrent systems can be reduced polynomially to the PCTLK model checking problem over concurrent probabilistic systems because all

CTLK formulae are PCTLK formulae as well (see Figure 5.1) and concurrent systems can be seen as concurrent probabilistic systems with all probability transitions equal to 1. Therefore, $MC(CTLK) \leq_P MC(PCTLK)$, so the result. □

5.4 Symbolic Representation with MTBDDs

In this section, we will introduce how to represent probabilistic interpreted systems with Multi-Terminal Binary Decision Diagrams (MTBDDs) and develop symbolic model checking algorithms for PCTLK. We choose MTBDD representation to be able to use the PRISM model checker [69] that provides an MTBDD engine. In fact, the MTBDD representations of probabilistic interpreted systems and MDPs, which are supported by PRISM, are exactly the same.

5.4.1 Introduction to MTBDDs

Binary Decision Diagrams (BDDs) were originally created by Lee[71] and enriched by Akers [1]. MTBDDs are extended from BDDs; thus they have inherent BDD features. An MTBDD M is a rooted directed acyclic graph associated with a set of ordered Boolean variables $x_1 < \cdots < x_n$. An MTBDD data structure can be considered as a mapping function $f_M(x_1, \ldots, x_n) : \mathbb{B}^n \to \mathbb{R}$. Nodes in MTBDDs are classed as either non-terminal or terminal. Unlike BDDs where terminal nodes can only be 0 or 1, terminal nodes in MTBDDs are allowed to be real numbers other than 0 and 1. A terminal node m is labeled by a real number $val(m)$. A non-terminal node m is labeled by a variable $var(m) \in (x_1, \ldots, x_n)$ and has $then(m)$ and $else(m)$ two children.

The order over the Boolean variables in MTBDDs is based on a position and value of a node. For two non-terminal nodes m_1 and m_2, if $var(m_1) \leq var(m_2)$,

then $m_1 \leq m_2$. A non-terminal node is less than a terminal node, for example, if m_1 is a non-terminal node and m_2 is a terminal node, then $m_1 < m_2$. The order for a non-terminal node m and its children $then(m)$ and $else(m)$ is $m < then(m)$ and $m < else(m)$. In MTBDDs' data structure, two terminal nodes m_1 and m_2 are equivalent if and only if $val(m_1) = val(m_2)$. For non-terminal nodes, two nodes are identical when conditions: 1) $var(m_1) = var(m_2)$, 2) $then(m_1) = then(m_2)$, and 3) $else(m_1) = else(m_2)$ are all satisfied. MTBDDs are efficient because they can be compact without any redundancy. For example, if two children are equal, $then(m_1) = else(m_2)$, then these two nodes will be combined to one node and all incoming edges are redirected to this unique child. MTBDDs have been used to encode real-valued matrices in [30]. These MTBDDs can be considered as a square $2^n \times 2^n$ matrix mapping function: $\mathbb{B}^n \times \mathbb{B}^n \to \mathbb{R}$. Every element a_{ij} can be viewed as the value of a function $f : \{0, \ldots, 2^{n-1}\} \times \{0, \ldots, 2^{n-1}\} \to \mathbb{R}$. If we take Boolean variables $(x_1, \ldots, x_n)$ to range over row indices and Boolean variables $(y_1, \ldots, y_n)$ to range over column indices, this matrix can be seen as a probability transition matrix of Markov chains; therefore an MTBDD can be used to represent the probabilistic transition system. We use an MTBDD obtained from $f : (x_1, y_1, \ldots, x_n, y_n)$, which means we require the variables from the rows and columns to alternate, so that a recursive structure on the matrix is suited for efficient recursive algorithms for all standard matrix operations [37].

5.4.2 MTBDDs Model Representation

Now let us encode a probabilistic interpreted model M_{PIS}. Given an $M_{PIS} = (W, \mathbf{P_t}, I_{init}, \sim_1, \ldots, \sim_n, AP, V)$, then the number of Boolean variables required to encode states W is $m = \lceil log_2|W| \rceil$, where $|W|$ is the number of states in the system. We use an m Boolean variable tuple $\overline{x} = (x_1, \ldots, x_m)$ to represent each element

$s \in W$. Each tuple $\overline{x} = (x_1, \ldots, x_m)$ is then identified with a Boolean formula, represented by a conjunction of variables or their negation. Consequently, the set of states is encoded by taking the disjunction of the Boolean formulae encoding the states. We also introduce m more variables $\overline{y} = (y_1, \ldots, y_m)$ to encode the destination states. If we encode Boolean variables $\overline{x}$ and $\overline{y}$ to range over source and destination states, we can encode the transition probability matrix $\mathbf{P_t}$ of M_{PIS} model [84]. The initial distribution I_{init} can be represented by a vector using real numbers. Each agent in the model can be encoded as a k Boolean variables $\overline{z} = (z_1, , \ldots, z_k)$, where $k = \lceil log_2(n+1) \rceil$ ($n+1$ is the number of agents in the system including the system itself). Then, we use the same method used for $\mathbf{P_t}$ encoding to encode n epistemic accessibility relations $\sim_i$. If there is an epistemic accessibility relation between a source state and a destination state for agent i, we set the matrix $\sim_i$ for elements that represent this source state and destination state value as 1. (For purposes of simplification, we omit the representations of the atomic propositions and labeling functions).

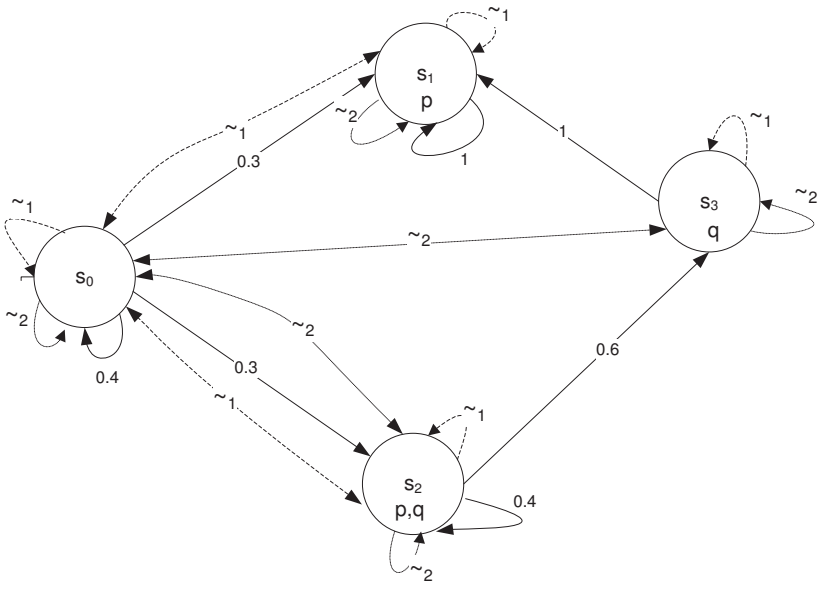

Figure 5.2: An Example of Probabilistic Interpreted System Model

We use an example to illustrate the encoding process. The probabilistic interpreted system is shown in Figure 5.2. There are two agents in the system. The solid lines with numbers are probabilistic transitions. The dashed lines with $\sim_1$ are epistemic accessibility relations for agent 1, while lines with $\sim_2$ represent relations for agent 2. The probability transition function $\mathbf{P_t}$, initial distribution and epistemic accessibility relations are listed in the following matrices. We normalize epistemic accessibility relations such that the probabilities from state s must sum to 1 and use $\approxeq$ to stand for normalization.

$$\mathbf{P_t} = \begin{bmatrix} 0.4 & 0.3 & 0.3 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0.4 & 0.6 \\ 0 & 1 & 0 & 0 \end{bmatrix} \qquad I_{init} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\sim_1 = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \approxeq \begin{bmatrix} 1/3 & 1/3 & 1/3 & 0 \\ 1/2 & 1/2 & 0 & 0 \\ 1/2 & 0 & 1/2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \sim_2 = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \approxeq \begin{bmatrix} 1/2 & 0 & 1/2 & 0 \\ 0 & 1 & 0 & 0 \\ 1/2 & 0 & 1/2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

To convert this probabilistic interpreted system into MTBDD, we need 2 Boolean variables for encoding agents $\overline{z} = (z_1, z_2)$ and 2 Boolean variables for encoding states $\overline{x} = (x_1, x_2)$. $z_1 = 0$ and $z_2 = 0$ represents the system, $z_1 = 0$ and $z_2 = 1$ is agent 1, and $z_1 = 1$ and $z_2 = 0$ stands for agent 2. Similarly for states: there are 4 states in the system; therefore we can use $(x_1, x_2) = (0, 0)$, $(x_1, x_2) = (0, 1)$, $(x_1, x_2) = (1, 0)$, and $(x_1, x_2) = (1, 1)$ to stand for s_0, s_1, s_2, and s_3.

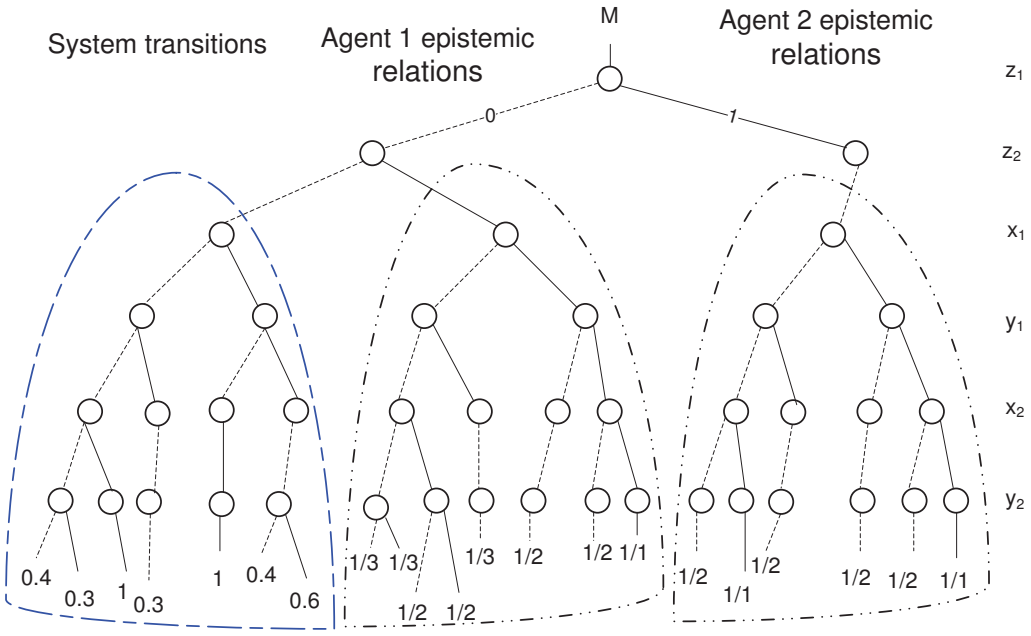

Figure 5.3: An MTBDD Encoding Example

Figure 5.3 is the MTBDD M for the probabilistic interpreted system. Non-terminal nodes are drawn as circles and terminal nodes as the numbers indicating the value of nodes. For the purpose of clarification, we omit the zero terminal node and any edges which lead directly to it . Non-terminal node *then* and *else* offsprings are connected with solid and dashed lines respectively to their parent. We divide this MTBDD into three parts: $(z_1, z_2) = (0, 0)$, $(z_1, z_2) = (0, 1)$ and $(z_1, z_2) = (1, 0)$ to represent probability transitions, epistemic relations for agent 1, and epistemic relations for agent 2.

We can trace the appropriate path through the MTBDD to get the value of the function f_M. Tabel 5.2 generates from Figure 5.3 and shows the function f_M which the MTBDD M represents. For example, $f_M(z_1, z_2, x_1, y_1, x_2, y_2) = f_M(0, 0, 1, 1, 0, 1) = 0.6$. We denote $(s, s')^0$ to represent probability transition from state s to s', $(s, s')^i$ to be epistemic accessibility relations for agent i. Based on our encoding rules, we know

Table 5.2: The Function that The MTBDD Represents

Agent Number	z_1	z_2	x_1	y_1	x_2	y_2	$f_M(z_1, z_2, x_1, y_1, x_2, y_2)$	$\mathbf{P_{IS}}^{\mathscr{A}}$
System	0	0	0	0	0	0	0.4	$(s_0, s_0)^{\mathscr{A}_0} = 0.4$
	0	0	0	0	0	1	0.3	$(s_0, s_1)^{\mathscr{A}_0} = 0.3$
	0	0	0	1	0	0	0.3	$(s_0, s_2)^{\mathscr{A}_0} = 0.3$
	0	0	0	0	1	1	1	$(s_1, s_1)^{\mathscr{A}_0} = 1$
	0	0	1	0	1	1	1	$(s_3, s_1)^{\mathscr{A}_0} = 1$
	0	0	1	1	0	0	0.4	$(s_2, s_2)^{\mathscr{A}_0} = 0.4$
	0	0	1	1	0	1	0.6	$(s_2, s_3)^{\mathscr{A}_0} = 0.6$
Agent 1	0	1	0	0	0	0	1/3	$(s_0, s_0)^{\mathscr{A}_1} = 1/3$
	0	1	0	0	0	1	1/3	$(s_0, s_1)^{\mathscr{A}_1} = 1/3$
	0	1	0	1	0	0	1/3	$(s_0, s_2)^{\mathscr{A}_1} = 1/3$
	0	1	0	0	1	0	1/2	$(s_1, s_0)^{\mathscr{A}_1} = 1/2$
	0	1	0	0	1	1	1/2	$(s_1, s_1)^{\mathscr{A}_1} = 1/2$
	0	1	1	0	0	0	1/2	$(s_2, s_0)^{\mathscr{A}_1} = 1/2$
	0	1	1	1	0	0	1/2	$(s_2, s_2)^{\mathscr{A}_1} = 1/2$
	0	1	1	1	1	1	1	$(s_3, s_3)^{\mathscr{A}_1} = 1$
Agent 2	1	0	0	0	0	0	1/2	$(s_0, s_0)^{\mathscr{A}_2} = 1/2$
	1	0	0	1	0	0	1/2	$(s_0, s_2)^{\mathscr{A}_2} = 1/2$
	1	0	0	0	1	1	1	$(s_1, s_1)^{\mathscr{A}_2} = 1$
	1	0	1	0	0	0	1/2	$(s_2, s_0)^{\mathscr{A}_2} = 1/2$
	1	0	1	1	0	0	1/2	$(s_2, s_2)^{\mathscr{A}_2} = 1/2$
	1	0	1	1	1	1	1	$(s_3, s_3)^{\mathscr{A}_2} = 1$

that $\mathbf{P_{IS}}(s_2, s_3)^0 = 0.6$. In fact, we can abstract the transition probability matrix $\mathbf{P_{IS}}$ of a concurrent probabilistic system. We use superscript to indicate adversaries.

$$
\mathbf{P_{IS}} = \begin{array}{c} \\ s_0;\mathscr{A}_0 \\ s_0;\mathscr{A}_1 \\ s_0;\mathscr{A}_2 \\ --- \\ s_1;\mathscr{A}_0 \\ s_1;\mathscr{A}_1 \\ s_1;\mathscr{A}_2 \\ --- \\ s_2;\mathscr{A}_0 \\ s_2;\mathscr{A}_1 \\ s_2;\mathscr{A}_2 \\ --- \\ s_3;\mathscr{A}_0 \\ s_3;\mathscr{A}_1 \\ s_3;\mathscr{A}_2 \end{array}
\begin{pmatrix}
0.4 & 0.3 & 0.3 & 0 \\
1/3 & 1/3 & 1/3 & 0 \\
1/2 & 0 & 1/2 & 0 \\
--- & --- & --- & --- \\
0 & 1 & 0 & 0 \\
1/2 & 1/2 & 0 & 0 \\
0 & 1 & 0 & 0 \\
--- & --- & --- & --- \\
0 & 0 & 0.4 & 0.6 \\
1/2 & 0 & 1/2 & 0 \\
1/2 & 0 & 1/2 & 0 \\
--- & --- & --- & --- \\
0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 \\
0 & 0 & 0 & 1
\end{pmatrix}
$$

The modified NetBill protocol is implemented with PRISM in the next section to verify probabilistic and epistemic properties to illustrate our approach.

5.5 Case Study

PRISM is a tool for the automatic verification of randomized or probabilistic behaviors via probabilistic model checking. PRISM allows us to verify PCTL formulae over Markov chains or Markov decision processes. In this section, we model the NetBill

protocol and verify its probabilistic and epistemic properties using this tool. Although PRISM does not support PCTLK logic, with our approach introduced in Section 5.2, we are able to convert PCTLK logic into PBTL logic with an existential quantifier. Moreover, we know from Section 2.4 that all PBTL formulae with an existential quantifier are the same as PCTL formulae. Therefore, with our model and formulae reduction, we can use the PRISM model checker to verify probabilistic epistemic properties.

5.5.1 Encoding

NetBill protocol is developed for online shopping encryption. The basic protocol involves one customer agent *Cus* and one merchant agent *Mer* interacting to finish an online shopping process. This protocol can also be applied to more than one customer and one merchant. The size of the model can be easily scaled up by the number of customers and merchants to benchmark the time and space complexity of our approach. We modified this protocol with probability transitions.

A customer requests a quote from the merchant for an item to initialize the protocol. Five percent (5%) of these requests will fail to be sent to the merchant due to internet connection issues. The merchant replies to the successfully delivered request by presenting a quote for the requested item. After a customer gets the quote, thirty percent (30%) of customers reject the offer and end the protocol without purchase. The other 70% of customers accept the offer and pay for the item. Ten percent (10%) of payments will be nullified due to connection issues and card issues. Thus, after the merchant provides the quote to the customer, there is 63% chance she will receive the payment for the item. When the merchant receives the payment, then she will deliver the items to the customer. Ninety-nine percent (99%) of deliveries are successful. If the delivery fails, the merchant will refund the customer. Figure 5.4

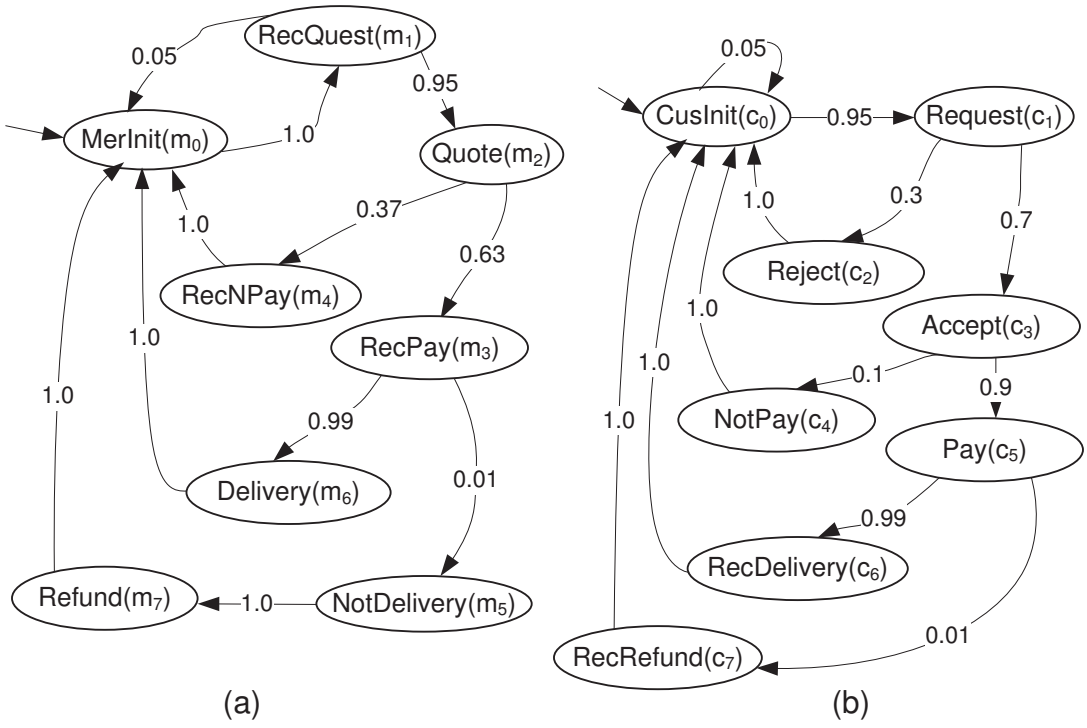

Figure 5.4: NetBill Protocol Probability Transition (a) Merchant, (b) Customer

depicts the merchant and customer probabilistic models.

We encode this protocol by using a *module* in PRISM for every agent and the entire system is defined as a process in which these *agent modules* can interact with each other. The local states of the merchant or the customer are defined as variables. The global states of the system are obtained from the Cartesian product of the local states of all modules. Therefore, the epistemic accessibility relations for agent i at state s include all the global states with *module* i's local variable s. We use labels to define the epistemic relations. To illustrate, label $A1s1$ includes a set of states that are equivalent for agent $A1$ at s_1 (λ_1). In other words, label $A1s1$ includes all states that show other agents' next-step behaviors when agent $A1$ is in state s_1.

```
label "A1s1" = (s1=0|s1=1);
```

Therefore, $A1s1$ is considered as the set of states that comprises all the states related

to s_1 by $\sim_1$.

The probability transitions of each *module* are described by a set of commands that indicate the updated local variables and probabilities for a given local variable. We uploaded the entire code for verifying the epistemic and probabilistic properties for the NetBill protocol using the PRISM probabilistic model checker on the open-source project website SourceForge.net[2]

Many properties, such as safety, liveness, and reachability, can be checked in the NetBill protocol and are easy to express using PCTLK. Some examples of tested formulae are listed in Table 5.3 (for a system including one merchant and one customer):

Table 5.3: Formula Examples

$\phi_1 = \neg Pay \rightarrow P_{\leq 0.02}(K_{mer} Pay)$
$\phi_2 = \neg P_{\geq 0.98}(F((K_{cus} Delivery) \wedge \neg Delivery))$
$\phi_3 = Delivery \rightarrow P_{\geq 0.99}(K_{cus} Deliver)$
$\phi_4 = P_{\geq 0.55} K_{cus} Deliver$

Formulae ϕ_1 and ϕ_2 express examples of the safety properties i.e., "unexpected events will not occur in the system". In conventional quality model checking, the safety property is expressed by $AG\neg p$, where p is an "unexpected" situation that should be avoided. However, human beings make mistakes in the real world; thus, when we design a system, we set a confidence interval such that we allow for inconsistencies to occur within a buffer zone. In probability model checking, safety properties can be expressed as "unexpected events" have a low probability of occurring or high probability of not occurring. For example, "chances that a merchant thinks that a customer has already paid but the customer has not actually paid is less than 2%" (formula ϕ_1), or "there is greater than 98% chance that it will not occur that the customer receives the item and the merchant did not deliver it to her" (formula ϕ_2).

[2]https://sourceforge.net/projects/mcepistemicprob/files/NetBill/?

After we define the epistemic relations as labels "CustomerNPay" and "Merchant-KnowPay", the same properties are expressed in PRISM as follows:

$$\text{“}CustomerNPay\text{”} \Rightarrow P <= 0.02\ [F\ (P >= 1[X\text{“}MerchantKnowPay\text{”}])] \quad (5.3)$$

$$P >= 0.98\ [F\ (P >= 1[X\text{“}CustomerKnowDel\text{”}) \wedge (!\text{“}MerchantDel\text{”})] \quad (5.4)$$

In contrast to the safety property, a liveness property states "an expected event will eventually happen". Formula ϕ_3 expresses the liveness property that after the merchant delivers the item, the customer will be aware of it and will receive the item eventually. The liveness property can be expressed in PRISM as follows:

$$\text{“}MerchantDel\text{”} \Rightarrow P >= 0.99\ [F\ (P >= 1[X\text{“}CustomerKnowDel\text{”})] \quad (5.5)$$

Formula ϕ_4 is an example of a reachability property, which means a particular situation can be reached from the initial state. In probabilistic model checking, we are able not only to state that a particular event will eventually occur, but also to present the probability that this event happens. If formula ϕ_4 is true, it means that there is a 55% chance a customer will receive a delivered item after having sent a request. The following expression in PRISM encodes this property.

$$P >= 0.55\ [F\ (P >= 1[X\text{“}MerchantDel\text{”})] \quad (5.6)$$

We have encoded the NetBill protocol with PRISM using the method presented in this section and we have verified this protocol experimentally with up to 15 agents. Experimental results are reported in the next section.

5.5.2 Experimental Results

The experimental results of the example described in the previous section are presented in Table 5.4. These results were performed using PRISM 4.1 on a Toshiba Portégé computer with 2.00 GHz Intel Core2 Duo T6400 processor and 3GB memory under a 64-bit Windows Vista Operating system.

Table 5.4: Experimental Results for Netbill Protocol with PRISM

No. of Agents	Model		MTBDD		Construction Time(sec)	Memory (MB)
	States	Transitions	Nodes	Leaves		
2	24	51	144	13	0.038	0.360
3	186	545	537	20	0.044	0.512
4	1,606	5,811	2,842	38	0.062	0.792
5	11,670	49,561	7,662	69	0.086	1.656
6	81,190	395,871	16,401	97	0.145	2.352
7	560,886	3,093,505	29,929	134	0.203	3.588
8	3,900,166	24,049,791	49,364	171	0.289	5.088
9	27,447,270	187,402,273	76,328	228	0.405	6.68
10	195,874,150	1,468,699,791	112,723	282	0.601	13.428
11	1,417,856,406	11,592,991,393	160,310	365	0.851	16.472
12	10,403,792,326	92,189,803,551	221,384	440	1.194	20.772
13	77,299,568,070	738,305,984,545	298,209	536	1.721	34.492
14	580,781,161,510	5,950,385,865,711	393,229	630	2.561	42.143
15	4,406,497,035,126	48,220,171,890,721	572,245	994	3.733	65.392

Table 5.4 reports the state space, MTBDD size, time and memory usage in the construction of the example. We increase the number of agents by adding more merchants or customers into the NetBill system. For reasons of simplicity, we have increased customer agents and kept a maximum of two merchants in the system. The second and third column report statistical data of the model (number of states and number of transitions) that reflect the state space. Column four and five present the number of nodes and leaves in MTBDD that represent the model. The last two columns indicate the time for converting the PRISM model into an MTBDD symbolic model and the memory usage for constructing the model. From Table 5.4, we notice that the number of states surges dramatically as the number of agents increases, but the growth is not exponential. Moreover, the size of the MTBDD data structure

(including nodes and leaves) grows much more slowly than the model size. This means that the MTBDD data structure is more efficient as the model size increases. The time and memory usage for constructing the model do not increase exponentially, but only polynomially when augmenting the number of agents (See Figure 5.5). In order to compare time and memory usage in the same figure, we modified this by dividing memory usage by 10. These experimental results confirm the complexity presented in Section 5.3.

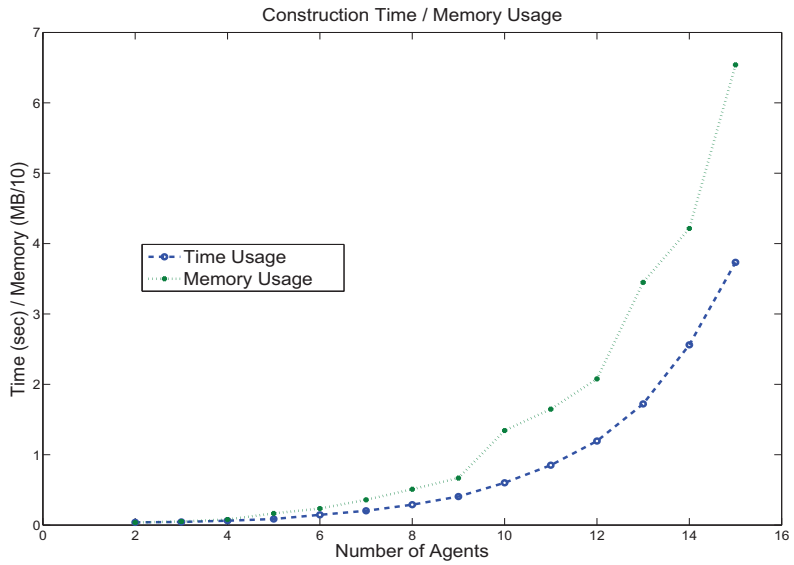

Figure 5.5: Model Construction Time/Memory Usage

We verified the four formulae listed in the previous section. All those formulae hold for a one-customer and one-merchant NetBill protocol. As all the formulae are similar in terms of time and space complexity, in the following, we only focus on the reachability property (formula ϕ_4). Table 5.5 shows verification results of this formula. Agents' types are indicated in this table as C and M for customer and merchant respectively. We also included the number of Boolean variables required to encode the NetBill protocol in column "Bool vars". We found that although PRISM is able to construct models for NetBill protocol with 15 agents, it only can verify

a formula for a model with 13 agents (2 merchants and 11 customers). There are 39 Boolean variables that are required to encode NetBill protocol with 13 agents, corresponding to a maximal state space of size $2^{39} \approx 10^{11}$. This result is consistent with PRISM specification for the MTBDD engine [3].

Table 5.5: Verification Results for Formula ϕ_4

Agents No.	Results	States	Transitions	Bool vars	Time(sec)	Memory (MB)
2 (1M1C)	True	24	51	6	0.007	1.184
3 (2M1C)	True	186	545	9	0.028	1.844
4 (2M2C)	True	1,606	5,811	12	0.146	2.924
5 (2M3C)	True	11,670	49,561	15	0.816	5.24
6 (2M4C)	True	81,190	395,871	18	3.626	20.068
7 (2M5C)	True	560,886	3,093,505	21	12.368	44.376
8 (2M6C)	False	3,900,166	24,049,791	24	39.691	67.892
9 (2M7C)	False	27,447,270	187,402,273	27	94.554	140.756
10 (2M8C)	False	195,874,150	1,468,699,791	30	226.196	175.356
11 (2M9C)	False	1,417,856,406	11,592,991,393	33	554.748	-
12 (2M10C)	False	10,403,792,326	92,189,803,551	36	1067.6	-
13 (2M11C)	False	77,299,568,070	738,305,984,545	39	2112.587	

Table 5.5 indicates that model checking time rises as the model gets larger. Memory usage follows the same trend as the verification time. Time and memory usage both increase polynomially with the number of agents and model size (see Figure 5.6). We also notice that the formula does not hold for 8 or more agents in the protocol (2 merchants and 6 customers in the trail). The reason is that as the number of agents in the system increases, interactive behavior among agents leads to increasing uncertainty.

We evaluate the probability of this property with "$P =?$" operator in PRISM. The result is listed in Table 5.6. The probability of the formula holding for two agents in the system is 0.6646, while for 8 agents the probability drops to 0.545. Figure 5.7 shows that after having 4 agents in the system, the probability of a customer knowing the item has been successfully delivered decreases linearly when the number of agents increases.

[3] http://www.prismmodelchecker.org/manual/FrequentlyAskedQuestions/PRISMModelling#max_model_size

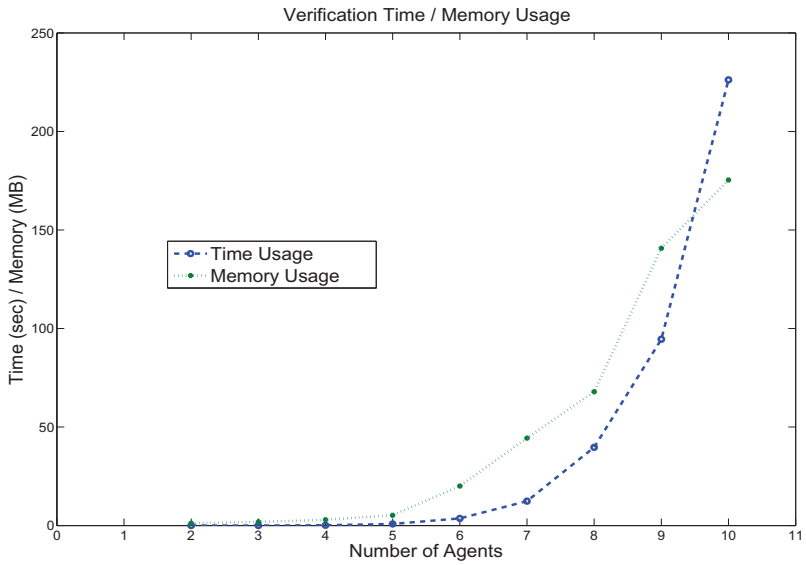

Figure 5.6: Verification Result of Formula ϕ_4

Table 5.6: Percentage of Successful Delivery

Agents No.	2	3	4	5	6	7
Probability	0.6646	0.6545	0.6518	0.629	0.6005	0.571
Agents No.	8	9	10	11	12	13
Probability	0.545	0.521	0.4996	0.4804	0.4633	0.4482

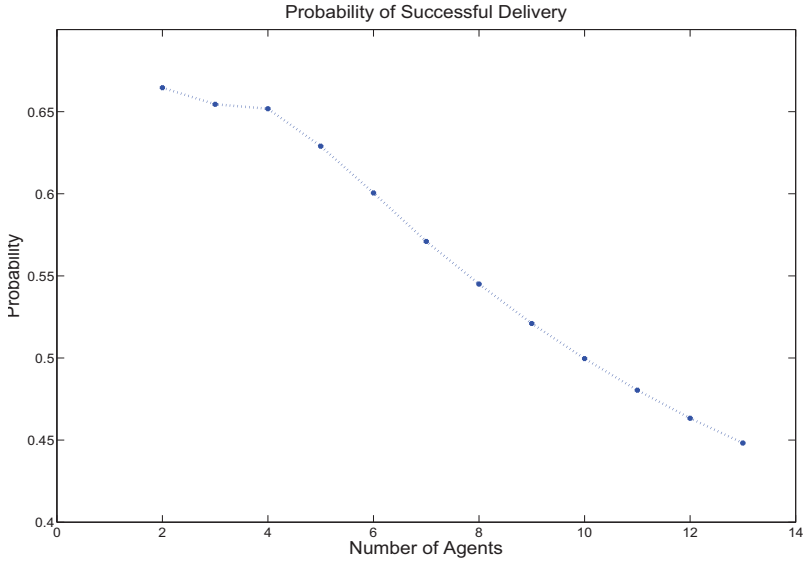

Figure 5.7: The Probability for a Successful Delivery

Chapter 6

Conclusions and Future Work

In this chapter, we first list the main contributions of our research in Section 6.1. The potential directions for future work are indicated in Section 6.2.

6.1 Contributions

The objective of this dissertation is to tackle the problem of epistemic-probabilistic model checking in MAS. The theoretical contributions are shown in Figure 6.1. The main contributions of our research may be summarized as follows:

- Theoretical contributions:

    1. We defined probabilistic-epistemic logic PCTLK [100, 101] to specify quantified knowledge properties in MAS. PCTLK not only allows probabilities of paths (i.e. runs), but also represents quantified knowledge. We also built the model M_{PIS} of PCTLK, which integrated Discrete-Time Markov Chains (DTMCs) with interpreted systems, to model MAS. The reason behind choosing DTMCs as our basic models is that DTMCs are widely used to model systems with probability information and are formal models

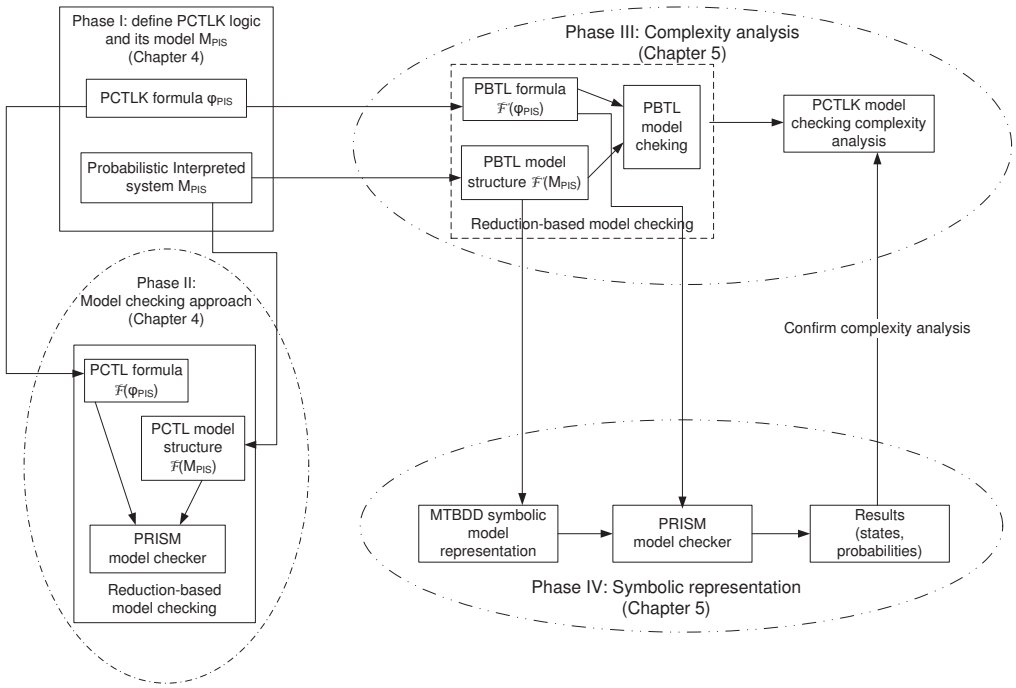

Figure 6.1: Theoretical Contributions

of PCTL, the predecessor of PCTLK. On the other hand, the formalism of interpreted systems has been proved to be an efficient formalism for modeling key characteristics of agents' knowledge to allow the inter-operability between global (the system) and local (agents) models. Therefore, M_{PIS} can effectively express quantified knowledge of MAS.

2. A new reduction-based model checking approach is developed to convert PCTLK model checking into PCTL model checking [102]. This reduction is achieved by transforming the models of PCTLK into MDPs, which are then transformed to DTMCs using the notion of *scheduler*. We proved the soundness and completeness of this transformation stating that a PCTLK formula is satisfied in a model of PCTLK if and only if a corresponding PCTL formula is satisfied in a DTMC model of PCTL. Therefore, formulae

of PCTLK can be simply checked using the probabilistic model checker PRISM.

3. The complexity of PCTLK model checking over concurrent probabilistic programs has been analyzed. We investigated the effectiveness of our approach along with analyzing the computational complexity of PCTLK model checking problem. We proved that PCTLK model checking in concurrent probabilistic programs is PSPACE-complete and can be solved in polynomial time in the size of the model and length of the formula for Markov chains.

4. We explored the symbolic representation of probabilistic interpreted systems with Multi-Terminal Binary Decision Diagrams (MTBDDs) to use the MTBDD engine of PRISM.

- Practical contributions: Two concrete case studies have been implemented with various model checkers to show applicabilities of our proposed techniques along with performance analysis and comparison with other approaches:

  1. A modified dining cryptographers protocol which includes uncertainty knowledge has been implemented with three different model checkers: the probabilistic model checker PRISM, the multi-agent model checker MCMAS, and the model checker for knowledge MCK. The results and comparisons are presented in [102].

  2. In [90] and [98], we verified probabilistic commitment properties and probabilistic knowledge of a modified NetBill protocol with PRISM. This protocol has also been used in [2] and is implemented using extended NuSMV and CWBNC to verify knowledge and commitment properties.

- Collaborations:

During my thesis research, I collaborated with colleagues in commitment and knowledge representations and jointly published a book chapter [11], a journal paper [90], two conference papers [43] and [42], and a journal paper is under review [2].

6.2 Future Work

Our probabilistic knowledge model checking is an ongoing project. There are a number of areas for future research in probabilistic and epistemic verification in MAS. In the future, we will focus on the following aspects:

- Currently, our model checking algorithms are based on reduction techniques to alleviate the state explosion problem of model checking. We plan to investigate the use of other model checking techniques such as symbolic model checking, unbounded model checking, predicate abstraction, etc. to find better solutions for probabilistic and epistemic model checking.

- Another important issue is to integrate the trust component to our probabilistic logic. An important application is to measure probability of agent's knowledge based on opinions of other agents in the systems and how much those agents are trusted.

- Model checking epistemic and probabilistic logic can also be applied to the knowledge revision area to check if by revision and addition of new knowledge, the model still satisfies specification properties. We consider the applicability of this model to a range of practical case studies in different areas such as services computing and business applications.

- We also like to develop dedicated tools for converting our probabilistic epistemic

model into models that are supported by existing model checker such as PRISM, MCK, MCMAS, and NuSMV.

- Developing a dedicated probabilistic model checker or embedding probabilistic epistemic features into the PRISM model checker are interesting problems for our future research. We are interested in collaborating with the PRISM team at Oxford to add our algorithms to a new version of PRISM.

Bibliography

[1] S.B. Akers. Binary decision diagrams. *Computers, IEEE Transactions on*, 100(6):509–516, 1978.

[2] F. Al-Saqqar, J. Bentahar, K. Sultan, and W. Wan. Model checking temporal knowledge and commitments in multi-agent systems using reduction. *Simulation Modelling Practice and Theory*, pages 45–68, 2015.

[3] R. Alur and T.A. Henzinger. Reactive modules. pages 207 – 218, Los Alamitos, CA, USA, 1996.

[4] I. Anciutti. A learning classifier system for emergent team behavior in real-time pomdp. In *2009 IEEE International Conference on Intelligent Computing and Intelligent Systems (ICIS 2009)*, volume vol.1, pages 733 – 738, Piscataway, NJ, USA, 2009.

[5] C. Baier, B. Haverkort, H. Hermanns, and J.-P. Katoen. Model-checking algorithms for continuous-time markov chains. *Software Engineering, IEEE Transactions on*, 29(6):524–541, 2003.

[6] C. Baier and J.-P. Katoen. *Principles of model checking*. MIT Press, Cambridge, Mass., 2008.

[7] C. Baier and M. Kwiatkowska. Model checking for a probabilistic branching time logic with fairness. *Distributed Computing*, 11(3):125–155, 1998.

[8] F. Belardinelli and A. Lomuscio. A complete first- order logics of knowledge and time. *AAAI Press*, pages 705–714, 2008.

[9] J. Bentahar, M. El-Menshawy, H. Qu, and R. Dssouli. Model checking communicative agent-based systems. *Knowledge-Based Systems*, 2012.

[10] J. Bentahar, J.J. Meyer, and W. Wan. Model checking communicative agent-based systems. *Knowledge-Based Systems*, 22(3):142–159, 2009.

[11] J. Bentahar, J.J. Meyer, and W. Wan. Model checking agent communication. In *Specification and Verification of Multi-agent Systems*, pages 67–102. Springer, 2010.

[12] O. Bernholtz, M. Y Vardi, and P. Wolper. An automata-theoretic approach to branching-time model checking. In *Computer Aided Verification*, pages 142–155. Springer, 1994.

[13] A. Bianco and L. De Alfaro. Model checking of probabilistic and nondeterministic systems. In *Foundations of Software Technology and Theoretical Computer Science*, pages 499–513. Springer, 1995.

[14] A. Biere, A. Cimatti, E. M. Clarke, O. Strichman, and Y. Zhu. Bounded model checking. *Advances in computers*, 58:117–148, 2003.

[15] A. Biere, A. Cimatti, E. M. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In Proc. of 5th TACAS*, Lectures Notes in Computer Science*, pages 193–207. Springer, 1999.

[16] P. Blackburn, M. De Rijke, and Y. Venema. Modal logic. *Cambridge Tracts in Theoretical Computer Science*, 53, 2001.

[17] R. H Bordini, M. Fisher, C. Pardavila, and M. Wooldridge. Model checking agentspeak. In *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pages 409–416. ACM, 2003.

[18] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *Computers, IEEE Transactions on*, 100(8):677–691, 1986.

[19] J. R. Burch, E. M. Clarke, and D. Long. Symbolic model checking with partitioned transition relations. *Computer Science Department*, page 435, 1991.

[20] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L.-J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and computation*, 98(2):142–170, 1992.

[21] S. Campos, E. M. Clarke, and M. Minea. The verus tool: A quantitative approach to the formal verification of real-time systems. In *Computer Aided Verification*, pages 452–455. Springer, 1997.

[22] Z. Cao. Model checking for epistemic and temporal properties of uncertain agents. *Agent Computing and Multi-Agent Systems*, pages 46–58, 2006.

[23] A. R. Cassandra, L. P. Kaelbling, and M. L. Littman. Acting optimally in partially observable stochastic domains. In *Proceedings of the National Conference on Artificial Intelligence*, volume 2, pages 1023–1023, Cambridge, MA, USA, 31,07-4,08 1995.

[24] K. Chatterjee, K. Sen, and T. A. Henzinger. Model-checking ω-regular properties of interval markov chains. In *Foundations of Software Science and Computational Structures*, pages 302–317. Springer, 2008.

[25] D. Chaum. The dining cryptographers problem: unconditional sender and recipient untraceability. *Journal of Cryptology*, 1(1):65–75, 1988.

[26] F. Ciesinski, C. Baier, M. Größer, and D. Parker. Generating compact mtbdd-representations from probmela specifications. In *Model Checking Software*, pages 60–76. Springer, 2008.

[27] A. Cimatti, E. M. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. Nusmv 2: An opensource tool for symbolic model checking. In *Computer Aided Verification*, pages 359–364. Springer, 2002.

[28] E. M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In *Logic of Programs, Workshop*, pages 52–71. Springer-Verlag, 1981.

[29] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 8(2):244–263, 1986.

[30] E. M. Clarke, M. Fujita, P. C. McGeer, K. L. McMillan, J. CY. Yang, and X. Zhao. Multi-terminal binary decision diagrams: An efficient data structure for matrix representation. 1993.

[31] E. M. Clarke, O. Grumberg, and D. E. Long. Model checking and abstraction. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 16(5):1512–1542, 1994.

[32] E. M. Clarke, O. Grumberg, and D. Peled. *Model checking*. MIT Press, Cambridge, Mass., 1999.

[33] R. Cleaveland and S. T. Sims. Generic tools for verifying concurrent systems. *Science of Computer Programming*, 42(1):39–47, 2002.

[34] M. Cohen, M. Dam, A. Lomuscio, and H. Qu. A symmetry reduction technique for model checking temporal-epistemic logic. In *IJCAI*, volume 9, pages 721–726, 2009.

[35] C. Courcoubetis and M. Yannakakis. Verifying temporal properties of finite-state probabilistic programs. In *Foundations of Computer Science, 1988., 29th Annual Symposium on*, pages 338–345. IEEE, 1988.

[36] C. Courcoubetis and M. Yannakakis. The complexity of probabilistic verification. *Journal of the ACM (JACM)*, 42(4):857–907, 1995.

[37] L. de Alfaro, M. Kwiatkowska, G. Norman, D. Parker, and R. Segala. Symbolic model checking of probabilistic processes using mtbdds and the kronecker representation. In *Tools and Algorithms for the Construction and Analysis of System: 6th InternationalConference,TACAS 2000*, volume 1785, page 395. Springer, 2000.

[38] M. I Dekhtyar, A. Dikovsky, and M. K. Valiev. Temporal verification of probabilistic multi-agent systems. In *Pillars of computer science*, pages 256–265. Springer, 2008.

[39] C. Delgado and M. Benevides. Verification of epistemic properties in probabilistic multi-agent systems. In *Proceeding of MATES 2009*, volume 5774 LNAI, pages 16–28, Hamburg, Germany, Sept. 9 - 11 2009. Springer Verlag.

[40] R. Durrett. *Probability: theory and examples*. Cambridge University Press, Cambridge; New York, 4th edition, 2010.

[41] M. El-Menshawy, J. Bentahar, W. El Kholy, and R. Dssouli. Reducing model checking commitments for agent communication to model checking arctl and gctl*. *Autonomous Agents and Multi-Agent Systems*, 27(3):375–418, 2013.

[42] M. El-Menshawy, J. Bentahar, W. Wan, and R. Dssouli. Verifying conformance of commitment protocols via symbolic model checking. *Proceedings of the International Workshop on Agent Communication*, pages 53–72, 2010.

[43] M. El-Menshawy, W. Wan, J. Bentahar, and R. Dssouli. Symbolic model checking for agent interactions. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 1555–1556, 2010.

[44] E. A. Emerson and E. M. Clarke. Using branching time temporal logic to synthesize synchronization skeletons. *Science of Computer programming*, 2(3):241–266, 1982.

[45] R. Fagin, J. Y. Halpern, and M. Y. Vardi. *Reasoning about knowledge*. MIT Press, Cambridge, 1995.

[46] M. Finger and D. M. Gabbay. Adding a temporal dimension to a logic system. *Journal of Logic, Language and Information*, 1(3):203–233, 1992.

[47] P. Gammie and R. Van der Meyden. MCK: model checking the logic of knowledge. In *Proc. of the 16th International Conference on Computer Aided Verification*, pages 479–483, Berlin, Germany, 13-17 July 2004. Springer-Verlag.

[48] H. Geffner and J. Wainer. Modeling action, knowledge and control. In *Proc. of European Conf. on AI (ECAI-98)*, pages 532–6, Chichester, UK, 23-28 Aug 1998. Wiley.

[49] C. M. Grinstead and J. L. Snell. *Introduction to probability.* American Mathematical Soc., 1998.

[50] J. Y. Halpern. *Reasoning about uncertainty.* MIT Press, Cambridge, Mass., 2003.

[51] J. Y. Halpern and M. Y. Vardo. Model checking vs. theorem proving: A manifestor. In *2nd International Conference on Principles of Knowledge Representation and Reasoning (KR'91)*, pages 325 – 334, San Mateo (CA), 1991.

[52] H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. *Formal Aspects of Comcputing*, 6(5):512–535, 1994.

[53] V. Hartonas-Garmhausen, S. Campos, and E. M. Clarke. Probverus: Probabilistic symbolic model checking. In *Formal Methods for Real-Time and Probabilistic Systems*, pages 96–110. Springer, 1999.

[54] T. Hérault, R. Lassaigne, F. Magniette, and S. Peyronnet. Approximate probabilistic model checking. In *Verification, Model Checking, and Abstract Interpretation*, pages 73–84. Springer, 2004.

[55] G. J. Holzmann. The model checker spin. *IEEE Transactions on software engineering*, 23(5):279–295, 1997.

[56] X. Huang, C. Luo, and R. Van Der Meyden. Symbolic model checking of probabilistic knowledge. In *13th Conference on Theoretical Aspects of Rationality*

and Knowledge, TARK 2011,, pages 177–186, Groningen, Netherlands, July 12 - 14 2011.

[57] X. Huang, K. Su, and C. Zhang. Probabilistic alternating-time temporal logic of incomplete information and synchronous perfect recall. In *Proceedings of AAAI*, 2012.

[58] G. E. Hughes and M. J. Cresswell. *A new introduction to modal logic*. Routledge, London; New York, 1996.

[59] W. Jamroga. A temporal logic for markov chains. In *Proceeding of 7th International Conference on AAMAS*, pages 697–704, Padgham, Parkes, May 12-16 2008.

[60] M. Kacprzak, A. Lomuscio, A. Niewiadomski, W. Penczek, F. Raimondi, and M. Szreter. Comparing BDD and SAT based techniques for model checking chaum's dining cryptographers protocol. *Fundamenta Informaticae (Netherlands)*, 72(1-3):215 – 34, 2006/07/.

[61] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra. Partially observable markov decision processes for artificial intelligence. In *Reasoning with Uncertainty in Robotics*, pages 146–62, Berlin, Germany, 4-6 Dec. 1995 1996. Springer-Verlag.

[62] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1-2):99–134, 05 1998.

[63] R. Kaplow, A. Atrash, and J. Pineau. Variable resolution decomposition for robotic navigation under a pomdp framework. In *Proceedings - IEEE International Conference on Robotics and Automation*, pages 369 – 376, 2010.

[64] S. Konur, M. Fisher, and S. Schewe. Combined model checking for temporal, probabilistic, and real-time logics. *Theoretical Computer Science*, 503:61–88, 2013.

[65] O. Kupferman, M. Y. Vardi, and P. Wolper. An automata-theoretic approach to branching-time model checking. *Journal of the ACM (JACM)*, 47(2):312–360, 2000.

[66] M. Kwiatkowska, G. Norman, and D. Parker. PRISM: probabilistic symbolic model checker. volume 2324 LNAI, pages 200 – 4, Berlin, Germany, 2002.

[67] M. Kwiatkowska, G. Norman, and D. Parker. Probabilistic symbolic model checking with prism a hybrid approach. *International Journal on Software Tools for Technology Transfer*, 6(2):128–42, 08 2004.

[68] M. Kwiatkowska, G. Norman, and D. Parker. Advances and challenges of probabilistic model checking. In *Communication, Control, and Computing (Allerton), 2010 48th Annual Allerton Conference on*, pages 1691–1698. IEEE, 2010.

[69] M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *Proc. 23rd International Conference on Computer Aided Verification (CAV'11)*, volume 6806 of *LNCS*, pages 585–591. Springer, 2011.

[70] M. Kwiatkowska, G. Norman, D. Parker, and R. Segala. Symbolic model checking of concurrent probabilistic systems using MTBDDs and Simplex. *School of computer science research reprots - University of Birmingham CSR*, 1999.

[71] C.-Y. Lee. Representation of switching circuits by binary-decision programs. *Bell System Technical Journal*, 38(4):985–999, 1959.

[72] A. Lomuscio, T. Łasica, and W. Penczek. Bounded model checking for interpreted systems: preliminary experimental results. In *Formal Approaches to Agent-Based Systems*, pages 115–125. Springer, 2003.

[73] A. Lomuscio, C. Pecheur, and F. Raimondi. Automatic verification of knowledge and time with nusmv. In *Proc. of the 20th Int. Conf. on AI (IJCAI07)*, pages 1384–1389, 2007.

[74] A. Lomuscio and W. Penczek. Symbolic model checking for temporal-epistemic logics. *SIGACT News*, 38(3):77–99, 2007.

[75] A. Lomuscio, H. Qu, and M. Solanki. Towards verifying contract regulated service composition. In *Web Services, 2008. ICWS'08. IEEE International Conference on*, pages 254–261. IEEE, 2008.

[76] A. Lomuscio, H. Y. Qu, and F. Raimondi. MCMAS: a model checker for multi-agent systems http://www-lai.doc.ic.ac.uk/mcmas/.

[77] A. Lomuscio and F. Raimondi. The complexity of model checking concurrent programs against ctlk specifications. In *Declarative Agent Languages and Technologies IV*, pages 29–42. Springer, 2006.

[78] A. Lomuscio and F. Raimondi. MCMAS: a model checker for multi-agent systems. In *Proc. of the 12th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 450–454, Berlin, Germany, 25 March-2 April 2006 2006. Springer-Verlag.

[79] J. Ma, G. Zhang, and J. Lu. A state-based knowledge representation approach for information logical inconsistency detection in warning systems. *Knowledge-Based Systems*, 23(2):125–131, 2010.

[80] K. L. McMillan. *Symbolic model checking*. Springer, 1993.

[81] K. L. McMillan. Applying sat methods in unbounded symbolic model checking. In *Computer Aided Verification*, pages 250–264. Springer, 2002.

[82] J. J. Meyer and W. van der Hoek. *Epistemic logic for AI and computer science*, volume 41. Cambridge University Press, Cambridge, 1995.

[83] S. Paquet, L. Tobin, and B. Chaib-draa. Real-time decision making for large POMDPs. volume 3501 LNAI, pages 450 – 455, Victoria, Canada, 2005.

[84] D. Parker. *Implementation of Symbolic Model Checking for Probabilistic Systems*. PhD thesis, University of Birmingham, 2002.

[85] W. Penczek and A. Lomuscio. Verifying epistemic properties of multi-agent systems via bounded model checking. *Fundamenta Informaticae*, 55(2):167–85, 05 2003.

[86] W. Penczek, B. Woźna, and A. Zbrzezny. Bounded model checking for the universal fragment of ctl. *Fundamenta Informaticae*, 51(1):135–156, 2002.

[87] A. S. Rao. Agentspeak(l): Bdi agents speak out in a logical computable language. In *Proceedings*, Agents Breaking Away. 7th European Workshop on Modelling Autonomous Agents in a Multi-Agent World, MAAMAW '96, pages 42–55, Berlin, Germany, 22-25 Jan. 1996 1996. Australian AI Inst., Melbourne, Vic., Australia, Springer-Verlag.

[88] P. Schnoebelen. The complexity of temporal logic model checking. *Advances in Modal Logic*, 4:393–436, 2002.

[89] G. Shani, R.I. Brafman, and S.E. Shimony. Model-based online learning of POMDPs. volume 3720 LNAI, pages 353 – 364, Berlin, Germany, 2005.

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

[91] Le T.-D., T. Komeda, and M. Takagi. Reinforcement learning for pomdp using state classification. In *2007 International Conference on Machine Learning; Models, Technologies & Applications (MLMTA 07)*, pages 45 – 51, Las Vegas, NV, USA, 2007.

[92] T. Taha, J.V. Miro, and G. Dissanayake. A pomdp framework for modelling human interaction with assistive robots. In *IEEE International Conference on Robotics and Automation*, pages 544 – 549, Piscataway, NJ, USA, 2011.

[93] R. van der Meyden and P. Gammie. MCK: model checking knowledge, http://www.cse.unsw.edu.au/m̃ck/.

[94] R. Van der Meyden and Kaile S. Symbolic model checking the knowledge of the dining cryptographers. In *Proceedings - 17th IEEE Computer Security Foundations Workshop, CSFW 04*, volume 17, pages 280–291, Pacific Grove, CA, United states, 2004 2004.

[95] R. Van der Meyden and N. Shilov. Model checking knowledge and time in systems with perfect recall. In *Proceedings 19th Conference on the Foundations of Software Technology and Theoretical Computer Science*, pages 432–445. Springer-Verlag, 13-15 Dec. 1999.

[96] M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proceedings of the First Symposium on Logic in Computer Science*. IEEE Computer Society, 1986.

[97] M. Y. Vardit. Automatic Verification of Probabilistic Concurrent Finite-State Programs. In *IEEE Symposium on Foundations of Computer Science*, pages 327–338, 1985.

[98] W. Wan, J. Bentahar, and A. Ben Hamza. On the complexity of model checking concurrent probabilistic programs against PCTLK specificiaiton. In *Submitted to Computers and Mathematics with Applications*.

[99] W. Wan, J. Bentahar, and A. Ben Hamza. Modeling and verifying agent-based communities of web services. In *Trends in Applied Intelligent Systems*, pages 418–427. Springer, 2010.

[100] W. Wan, J. Bentahar, and A. Ben Hamza. Model checking epistemic and probabilistic properties of multi-agent systems. In *Modern Approaches in Applied Intelligence*, volume 6704 LNAI, pages 68 – 78, Syracuse, NY, United states, 2011.

[101] W. Wan, J. Bentahar, and A. Ben Hamza. Quantitative model checking of knowledge. In *New Trends in Software Methodologies, Tools and Techniques: Proceedings of the Eleventh SoMeT_12*, pages 91–107, 2012.

[102] W. Wan, J. Bentahar, and A. Ben Hamza. Model checking epistemic-probabilistic logic using probabilistic interpreted systems. *Knowledge-Based Systems 50*, pages 279 – 295, 2013.

[103] F. Z. Wang and M. Kwiatkowska. An MTBDD-based implementation of forward reachability for probabilistic timed automata. In *Automated Technology for Verification and Analysis*, pages 385–399. Springer, 2005.

[104] M. Wooldridge. *An introduction to multiagent systems*. John Wiley & Sons, Chichester, UK., 2 edition, 2009.

[105] M. Wooldridge, M. Fisher, M.-P. Huget, and S. Parsons. Model checking multi-agent systems with MABLE. In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 2*, pages 952–959. ACM, 2002.

[106] D. Z. Zhang, R. Cleaveland, and E. W. Stark. The integrated CWB-NC/PIOA tool for functional verification and performance analysis of concurrent systems. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 431–436. Springer, 2003.

[107] C. H. Zhou, B. Sun, and Z. F. Liu. Abstraction for model checking the probabilistic temporal logic of knowledge. In *Artificial Intelligence and Computational Intelligence*, pages 209–221. Springer, 2010.