Multi-Valued Model Checking IoT and Intelligent Systems with Trust and Commitment Protocols

Ghalya Alwhishi

A Thesis

In the Concordia Institute

for

Information Systems Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy (Information Systems Engineering) at

Concordia University

Montréal, Québec, Canada

April 2024

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to certify that the thesis prepared

By:	**Ghalya Alwhishi**
Entitled:	**Multi-Valued Model Checking IoT and Intelligent Systems with Trust and Commitment Protocols**

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

Doctor of Philosophy (Information Systems Engineering)

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

Signed by the Final Examining Committee:

Dr. Fariborz Haghighat	Chair
Dr. Raphaël Khoury	External Examiner
Dr. Juergen Rilling	Internal Program Examiner
Dr. Roch Glitho	Examiner
Dr. Rachida Dssouli	Examiner
Dr. Jamal Bentahar	Supervisor

Approved by ______________________

Dr. Jun Yan, Graduate Program Director

February 8, 2024

Date of Defense

Dr. Mourad Debbabi , Dean

Faculty of Engineering and Computer Science

Abstract

Multi-Valued Model Checking IoT and Intelligent Systems with Trust and Commitment Protocols

Ghalya Alwhishi, Ph.D.

Concordia University, 2024

In the era of connectivity, numerous domains utilize multi-sensor Internet of Things (IoT) and Intelligent Systems (IS) applications, which involve complex interactions among numerous components in open environments. This complexity challenges the verification of these systems' reliability and efficiency. This study pioneers the verification of IoT applications and intelligent systems within multi-source data environments, employing multi-agent commitment and trust protocols, particularly in uncertain and inconsistent settings.

Our research introduces efficient frameworks to model and verify these systems, incorporating commitment and trust protocols in settings characterized by uncertainty and inconsistency. We extend existing logics of commitment CTL^{cc} and CTL^{c} and the logic of trust TCTL to multi-valued cases for reasoning about uncertainty and inconsistency. We introduce 3v-CTL^{c} and 3v-CTL^{cc}, three-valued logics of commitment for reasoning about uncertainty, and 4v-CTL^{c} and 4v-CTL^{cc}, four-valued logics of commitment for reasoning about inconsistency. In the context of trust, we introduce 3v-TCTL and 4v-TCTL, multi-valued logics for reasoning about uncertainty and inconsistency over systems with trust protocols.

To address the complexity and time needed for developing direct algorithms, coupled with the scarcity of multi-valued model checking tools, we developed reduction algorithms. These algorithms transform the introduced multi-valued logics of commitment and trust into their classical case or into CTL, facilitating interaction with efficient model checkers such as MCMAS+ and MCMAS^{t}, and NuSMV, respectively.

To demonstrate the practicality and applicability of the tool in real settings, we presented and reported experimental results over multiple IoT and IS applications in healthcare, finance, and smart buildings. Our findings indicate that the proposed approaches and the MV-Checker tool are highly efficient and scalable, providing accurate results under varying conditions.

Acknowledgments

First and foremost, I offer my deepest thanks to Allah Almighty, who said in the Holy Book: (وَعَلَّمَكَ مَا لَمْ تَكُن تَعْلَمُ ۚ وَكَانَ فَضْلُ اللَّهِ عَلَيْكَ عَظِيمًا) (*"And He taught you that which you knew not, and the bounty of Allah to you has been great."*).

At the end of this stage in my research journey, I find myself overwhelmed with gratitude for the unwavering support, invaluable guidance and feedback provided by my supervisor, Dr. Jamal Bentahar, during my research work. I am thankful for his belief in me and for introducing me to the attractive and challenging topic of Multi-valued Model Checking. His boundless expertise has enriched this research with valuable contributions.

I am indebted to my esteemed research committee members Dr. Rachida Dssouli, Dr. Roch Glitho, and Dr.Olga Ormandjieva, for their invaluable guidance and support. Their expertise and guidance greatly enhanced my research.

Special thanks to the Ministry of Higher Education, Libya, and Concordia University for the financial support. Big thanks to my father-in-law for following up on my study file.

I want to thank my colleagues Dr. Nagat Drawel for her great support in the early stages of my study and Narges Bahrloo for sharing her research interest and valuable insights. I am grateful to the researchers whose valuable work formed the foundation of my research, Amine Laarj, Dr. Chechik, Dr. Elkholy and Dr. Drawel. To my dear brother, Dr. Ahmed Elwhishi, your support during all the phases of my study has been invaluable.

For their endless encouragement, my heartfelt thanks to my parents, brothers, and sisters. An extraordinary thanks, filled with love, to my husband, Najeh, and my children, Batool, Adam, and Ibrahim. To my little ones, Ahmed and Elias, more playtime awaits, and rest assured that when you wake up at night, I'll be by your side, not at my desk. Love you all!

To my late father's soul, my mother and my family,

with eternally love and gratitude.

To my teachers who taught me throughout my studies from primary school to doctorate, with my deepest thanks and appreciation.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter covers:

- Context of Research
- Motivations
- Problems and Research Questions
- Objectives
- Contributions
- Overview of the Dissertation

1.1 Context of Research

As the research focuses on the agent's communications in IS and IoT, we start by providing a comprehensive understanding of the interactions and behaviors within IoT and IS systems. We commence by precisely delineating IoT and IS, illuminating their distinctive features and functionalities. Then, we explain how IoT and IS are considered multi-agent systems (MAS).

1.1.1 The Internet of Things (IoT)

The Internet of Things (IoT) [72, 56, 65] refers to a network of interconnected devices or things that can communicate and share data over the Internet. These devices can range from everyday objects like smartphones, watches, and home appliances to more specialized equipment in industries like healthcare, transportation, and manufacturing. An IoT system typically consists of the following components:

- Things or Devices: These are the physical objects or devices that are equipped with sensors, actuators, and communication hardware to collect and transmit data. Examples include sensors in a smart thermostat, a fitness tracker, or a smart refrigerator.
- Sensors and Actuators: Sensors gather data from the environment, such as temperature, humidity, light, motion, etc. Actuators, on the other hand, can perform actions based on received data, like adjusting the temperature, locking a door, or turning on a light.
- Connectivity: This refers to the network that allows the devices to communicate with each other and with external systems. This can be achieved through Wi-Fi, Bluetooth, cellular networks, or other wireless technologies.
- Data Processing and Storage: Once the data is collected, it needs to be processed and sometimes stored for further analysis or action. This can be done on the device itself, on a local gateway, or in the cloud.
- User Interface: This is how users interact with the IoT system. It can be through a mobile app, a web interface, voice commands, or other means.
- Applications and Services: These are the software applications or services that provide the logic and functionality for the IoT system. They may analyze the data, trigger actions, or provide insights based on the collected information.
- Security and Privacy: IoT systems need to have robust security measures in place to protect the data and ensure the integrity of the devices. This includes authentication, encryption, and secure update mechanisms.

- Analytics and Machine Learning: These technologies can be used to extract meaningful insights from the vast amounts of data generated by IoT devices. This can lead to improved decision-making and automation.

IoT technology has proved its efficiency and importance in different domains. For instance, IoT in healthcare has the potential to revolutionize patient care, enhance healthcare delivery, and improve outcomes [68, 1]. In the supply chain, IoT is revolutionizing supply chain management by providing unprecedented visibility and control over the entire process. It enables businesses to make data-driven decisions, reduce costs, improve efficiency, and enhance customer satisfaction.[69, 63]

1.1.2 Intelligent Systems (IS)

Intelligent Systems (IS) refers to computer-based systems that are designed to mimic human-like intelligence and perform tasks that typically require human intelligence. These systems are capable of learning, reasoning, problem-solving, and making decisions based on data and algorithms [48, 16]. Below are some of the key components and characteristics of IS:

- Learning Capability: Intelligent systems have the ability to learn from data. This can be through supervised learning, where they are trained on labeled examples, or through unsupervised learning, where they discover patterns and relationships in data without explicit guidance.
- Reasoning and Problem-Solving: These systems can analyze information, make inferences, and solve complex problems. They can follow logical rules and make decisions based on available information.
- Adaptability: Intelligent systems are designed to adapt and evolve over time. They can adjust their behavior based on changing circumstances or new data.
- Autonomy: Intelligent systems can operate with a degree of autonomy, making decisions and taking actions without constant human intervention.

- Data-Driven Decision Making: Intelligent systems rely heavily on data to make informed decisions. They can process large volumes of data to identify patterns, trends, and anomalies.

IoT and IS are Multi-agent Systems

Multi-Agent Systems (MAS) is a field of artificial intelligence and computer science that deals with systems composed of multiple interacting intelligent agents. An agent refers to a computational entity or software program that can perceive its environment, reason about it, make decisions, and take actions to achieve specific goals [83, 61].

The Internet of Things (IoT) and Information Systems (IS) can be considered Multi-Agent Systems based on the following:

IoT as a Multi-Agent System:

- Agents (Things): In an IoT environment, the "things" are the agents. These can be any physical device with sensors and actuators, such as smartphones, smart home appliances, industrial machines, etc.
- Interaction and Communication: IoT devices interact with each other and with humans through various communication protocols (like Wi-Fi, Bluetooth, Zigbee, etc.). They can share information, receive commands, and coordinate actions.
- Autonomous Decision-Making: IoT devices often have some level of decision-making capabilities. For example, a smart thermostat can autonomously adjust the temperature based on the preferences and detected occupancy.
- Goal-Oriented Behavior: Each IoT device is designed with specific goals or tasks in mind. For instance, a smart fridge's goal might be to maintain a certain temperature and notify when supplies are running low.

(IS) as a Multi-Agent System:

An intelligent system can be considered a multi-agent system (MAS) when it is designed to consist of multiple autonomous agents that interact with each other and the environment to achieve specific goals.

In both cases, the key elements that define IoT and IS as Multi-Agent Systems include agents, interaction and communication, autonomous decision-making, and goal-oriented behavior when multiple agents interact.

Agents in IoT and IS

In an IoT context, agents are software entities that perform tasks autonomously, interact with other agents or devices, and make decisions based on information they gather [80, 75]. Here are examples of the elements that can play the role of an agent in an IoT system:

- Sensor Agents: These agents collect data from sensors deployed in the IoT network. They gather information on various environmental parameters such as temperature, humidity, light levels, etc.
- Actuator Agents: Actuator agents receive instructions or commands from other agents or systems and take actions in the physical world. For example, they might control actuators like motors, valves, or switches.
- Communication Agents: Communication agents facilitate the exchange of information between different devices, sensors, actuators, or other agents in the IoT network. They handle protocols and data formatting and ensure reliable data transmission.
- Decision-Making Agents: These agents are responsible for making decisions based on the information they receive. They may use predefined rules, machine learning models, or other algorithms to determine appropriate actions.
- Security Agents: These agents focus on ensuring the security and integrity of data and communications within the IoT network. They may handle tasks such as authentication, encryption, and access control.
- User Interface Agents: These agents provide a means for users or other systems to interact with the IoT network. They might handle tasks like displaying information, receiving user inputs, or providing notifications.

In the context of intelligent systems, an “agent” refers to a component or entity that is capable of perceiving its environment, making decisions, and taking actions to achieve specific goals or objectives [70]. Here are examples of elements that can play the role of an agent in an intelligent system:

- Perception Agents: Responsible for collecting data and perceiving information from various sources. For example, image recognition software that processes visual data to identify objects.

- Communication Agents: Facilitate the exchange of information between different components or entities within the intelligent system. For example, message-passing protocols in a distributed computing system.

- Execution Agents: Responsible for executing actions or tasks based on instructions or commands. For example, robotic arms in a manufacturing plant perform specific tasks like welding or assembling.

- Decision-Making Agents: Make decisions based on available information and predefined rules or algorithms. For example, autonomous vehicles make decisions on navigation, speed, and lane changes based on sensor data.

In fact, IoT and IS concepts are often integrated. For instance, an intelligent system might be a component of an IoT system, providing the decision-making and intelligence layer for processing the data collected by IoT devices [62].

1.1.3 Trust and Commitment in IoT/IS

Trust and commitment protocols play a pivotal role in ensuring the reliability and security of IoT/IS systems. The main role of commitment and trust in such systems lies in providing social control that regulates the interactions and relationships among agents. Many formalisms and approaches that provide social modelling for commitments and trust in open systems have been introduced and can be found in literature, [38, 39, 32, 40, 41, 34, 37, 35] for commitment and [82, 22, 26, 21, 23, 29, 24, 28, 25, 27] for trust. For example, in [27], trust is modeled from a high-level abstraction based on the social behaviors of agents, and in [35], social commitments are modeled as agreements between two intelligent components, and they result from communicative actions between their interactions.

1.1.4 Model Checking Trust and Commitment Systems

Model checking systems involving social commitments or trust remains a significant ongoing challenge across various research domains. This technique generally models the system as a finite state machine and expresses the formula in the form of a temporal logic formula. Then, it verifies the model against the formula using a model checker and gives a verification answer: True (the system satisfied the formula) or False (the system satisfied the formula). Employing model checking proves to be an effective technique in verifying such systems and has been used in diverse domains.

In the context of trust, notable efforts have been made. For instance, in [12], an approach was introduced to model and verify trust models specified in a Colored Petri Net (CPN). This work presented TCPN, a new modeling formalism enabling evaluating such models from simulation and model checking perspectives. However, it did not offer a dedicated model checking technique for verifying trust-based models.

The work in [42] presented a model checking framework specifically designed to verify trust-based models against both regular and non-regular specifications. The authors modeled their system by introducing an algorithm generating a configuration graph of deterministic pushdown automata (PDA). They employed observation sequences to capture trust behaviors related to interactions between services and users. Nevertheless, this approach lacked formal semantics for trust, relying instead on trust pattern languages' context-free grammar to infer trust formulae. Furthermore, it proved incapable of formalizing and verifying trust-based autonomous MASs (Multi-Agent Systems).

Recent contributions addressed these limitations by providing efficient frameworks for modeling and verifying trust-based systems through direct and reduction model checking techniques. In [22, 29], authors introduced Trust Computation Tree Logic TCTL, a novel logic for trust incorporating preconditions. They also developed a transformation-based algorithm implemented in a Java toolkit that interacts automatically with the NuSMV model checker. Another proposal in [24] offered a formal model checking technique focusing on reasoning about trust relationships among groups of agents and other agents in multi-agent systems. This technique relied on a newly introduced logic named branching-time temporal logic (BT). In [26], authors introduced a model checking approach for assessing degrees of trust among agents in multi-agent systems, introducing a novel logic named TCTL^G.

In the context of commitments, model checking techniques have been instrumental in validating commitments-based systems. For instance, in [20], authors developed a framework that extends OWL-P, a language and toolset for protocols, into an ontology expressed in the Web Ontology Language (OWL). This extension supports the specification and composition of business protocols entailing commitments. Their framework facilitated model-checking composite protocols using the SPIN model checker.

In [35], a unified semantic model for social commitments and their operations was introduced. They proposed CTL^{*sc}, a logic extends CTL^* by incorporating commitment modalities alongside a novel definition for assignment and delegation operations. Demonstrating their model's efficacy, they showcased automatic verification using NuSMV and MCMAS symbolic model checkers, utilizing the NetBill protocol as an illustrative case.

Authors in [36] proposed a new logic called CTLC, by extending the temporal logic CTL with a new modality for social commitments. The proposed approach has shown that the problem of model checking CTLC can be reduced to the problems of model checking CTLK (a logic of time and knowledge) or ARCT (Action Restricted CTL). The authors proved the efficiency of their approach by implementing a verification of Contract Net Protocol modeled in terms of commitments and associated actions using NuSMV and MCMAS model checkers. Later in [38], authors modified their CTLC, a temporal logic of commitments for agent communication by introducing a new logic called CTL^+ for reasoning about communicating commitments and their fulfilment. Moreover, the authors introduced a reduction-based verification technique to reduce the problem of model checking CTL^+ into the problem of model checking ARCTL (the combination of CTL with action formulae) and the problem of model checking GCTL* (a generalized version of CTL* with action formulae) to use the extended NuSMV and the CWB-NC automata-based model checkers.

In [33, 31], authors proposed a new logic named CTL^{cc}, which extends CTL with modalities to represent conditional commitments and their fulfilment over the formalism of interpreted systems. They presented weak (strong) conditional commitment modalities to capture the different interactions between agents. The difference between these two modalities (weak and strong) is that strong conditional commitments are only established when there is a possibility to satisfy their conditions, while weak conditional commitments can be established even if the condition is never satisfied. The authors of this work analyzed the computational complexity of their approach and conducted the full implementation on top of the MCMAS model checker. Latter in [32], CTL^{cc} was expanded to encompass additional actions related to multiagent-based web services, like cancel and delegate, broadening its applicability to diverse business scenarios.

1.1.5 Multi-Valued Model Checking

As discussed earlier, the proposed classical model checking techniques generate only True or False answers. These techniques, in many cases, are insufficient to express and verify models' behaviors designed under multiple data source settings. In particular, in the case of uncertainty, inconsistency and multiple degrees of importance. Therefore, model checking techniques with multiple logics are needed to capture these settings. Multi-valued model checking has been proposed in several studies [17, 76, 77, 57, 50, 49, 78, 73, 66]. In [17], the authors proposed a direct multi-valued symbolic model checking technique to reason about uncertainty and inconsistency in the atomic propositions assigned to the states of the system and in the transitions between these states. They proposed a new logic called mv-CTL by extending the logic of CTL by multi-valued modalities that mainly rely on *mutli-valued sets* and *multi-valued relations*. This technique takes as inputs a multi-valued model and formulae and generates satisfaction degrees (T, M, F) or (TT, FF, FT, TF). The value M refers to "Maybe" as uncertain information. The values FT and TF refer to the inconsistency or conflict viewpoints between two designers about transitions between states or atomic propositions in particular system states. The values TT and FF refer to the agreement between the designers about the system transitions and atomic propositions. Moreover, the authors developed a multi-valued model checker called $\chi Check$ and applied their approach using this tool. The work in [13] introduced a model checking partial state spaces with 3-valued temporal logics to reason about partial or incomplete systems. The main idea behind this technique is to add the value $\perp$, which represents the incomplete information in the partial state space under investigation. The authors proposed a reduction algorithm which generates optimistic and pessimistic cuts from the three-valued system. The optimistic cut considers the value $\perp$ as "True", and the pessimistic cut considers the same value as "False". If the verification of the optimistic cut yields F, then the result is "False". If not, then check the pessimistic cut. If the latter yields "T" then the result is "True". If not, the result is "F", and this means we have True and False from both cuts, which confirms the uncertainty. In [53], the authors proposed a multi-valued version of (CTL*) named (mv-CTL*). In this logic, both the propositions and the transitions are taking values over a finite quasi-Boolean algebra. To implement the proposed method, a reduction approach was produced to reduce mv-CTL* to CTL*. In [15], Bruns and Godefroid show how to reduce multi-valued model checking with any distributive DeMorgan lattice to two-valued model checking. The logic used in this work is the modal mu-calculus, which is a propositional modal logic extended with Fixed-point operators [54]. The idea behind this technique is to use the distributive lattice L to define a set of "experts" with the same number of the join-irreducible elements in the lattice and then to derive from the multi-valued Kripke structure a standard Kripke structure for each expert to recall an existing model checker tool. Recently, in [58], the authors proposed their method, computation tree logic model checking based on multi-valued possibility measures. They modeled the multi-valued logic systems by multi-valued Kripke structures and introduced a new logic named (MvCTL) based on generalized possibility and necessity measures. This logic is more general than mv-CTL and addresses some of the latter's deficiencies but cannot be reduced to the classical CTL. In [50], the authors introduced a multi-valued logic named multi-valued alternating-time temporal logic (mv-ATL$^*_\rightarrow$) by extending the logic of ATL* to specify strategic abilities in multi-agent systems. The proposed approach is a reduction-based model checking that enables the use of the existing model checkers.

In summary, we compare the aforementioned existing approaches by taking into consideration the following criteria: **Formalization**, **Underlying Modeling Framework**, **Explicit Notion of Trust**, **Explicit Notion of Commitment**, **Verification Method**, **Use of Multi-logics**, **Applicability for Model Checking**, **Applicability for Multi-valued Model Checking**.

Formalization reflects using formal logics such as TCTL, CTLC and mv-CTL to represent and specify the commitments and trust. The **Underlying Modeling Framework** reflects the underlying models used for modeling the systems. **Explicit Notion of Trust** and **Explicit Notion of Commitment** show the possibility of expressing trust or commitments by explicit modalities. The **Verification Method** reflects the use of a formal verification technique to verify the proposed approach. The **Use of Multi-logics** shows if the underlying logic of the proposed approach is presented in the form of multi-valued logic that considers more than True or False truth values. Finally, **Applicability for Model Checking** and **Applicability for Multi-valued Model Checking** reflect the applicability of these two techniques for verifying the proposed approach.

Table 1.1 compares the existing approaches based on the aforementioned criteria. The table clearly demonstrates the limitations of the existing trust and commitment-based verification approaches in providing practical and formal frameworks that handle multi-valued cases. The table is divided into two parts representing our review: The first part is for model checking the two common multi-agent systems protocols, trust and commitment; the second part is for multi-valued model checking systems with uncertainty and inconsistency without involving trust or commitments.

More specifically, the last two columns of the first part of the table show the gap between these approaches and the use of multi-valued logic, which is essential for reasoning about important concepts such as uncertainty and inconsistency in trust and commitment-based systems. Inversely, The fourth and the fifth columns of the second part show the gap between the multi-valued approaches and the use of commitment and trust notions. Our

Table 1.1: Comparison between the list of publications reviewed for this proposal with respect to the proposed criteria

Approach	Formal	Underlying Modeling Framework	Notion of Trust	Notion of Commitments	Verification Method	Applicable for Model Checking	Multi-logic	Applicable for mv-Model Checking
Bidgoly et al. [12]		Colored Petri Net (CPN)	✓	—		—	—	—
El-Qurna et al. [42]	✓	Deterministic pushdown automata (PDA)	✓	—	FLC model checking	✓	—	—
Drawel et al. [22, 29]	✓	Vector-based interpreted systems	✓	—	Model Checking $TCTL$	✓	—	—
Drawel et al. [26]	✓	Vector-based interpreted systems	✓	—	Model Checking $TCTL^G$	✓	—	—
Drawel et al. [24]	✓	Vector-based interpreted systems	✓	—	Model checking BT	✓	—	—
Desai et al. [20]	✓	Web Ontology Language (OWL)	—	✓	Model Checking Commitment Protocols and their Compositions	✓	—	—
El-Menshawy et al. [35]	✓	Kripke-like structure	—	✓	Model checking CTL^{*sc}	✓	—	—
El-Menshawy et al. [36]	✓	Interpreted system	—	✓	Model Checking CTLC	✓	—	—
El-Menshawy et al.[38]	✓	Interpreted system	—	✓	Model checking $CTLC^+$	✓	—	—
E. Kholy et al. [31, 38]	✓	Interpreted system	—	✓	Model checking $CTLC^{cc}$	✓	—	—
Chechik et al. [17]	✓	Multi-valued Kripke structure	—	—	Multi-valued Model Checking	—	✓	✓
Bruns et al. [13]	✓	Partial Kripke structure	—	—	Model Checking 3-Valued CTL	✓	✓	✓
Konikowska et al [53]	✓	Multi-valued finite state model	—	—	Model checking for $mv-CTL^*$	✓	✓	✓
Bruns et al. [15]	✓	Multi-valued Kripke structure	—	—	Multi-valued model checking μL	✓	✓	✓
Y.Li et al. [58]	✓	Multi-valued Kripke structure	—	—	Model checking mv-CTL	—	✓	✓
Jamroga et al. [50]	✓	Multi-valued concurrent game structure (CGS)	—	—	Model Checking $ATL^*_{\to}$	✓	✓	✓

work fills these gaps in this research.

1.2 Motivations

Although model checking techniques have proven their efficiency in verifying systems with different communication protocols [81, 27, 31, 45], they still face significant challenges in verifying multi-source data systems under uncertain or inconsistent settings, especially when multi-agent commitment and trust protocols are involved. More specifically, the existing model checking techniques for verifying the commitment and trust systems give only absolute (*True*) or (*False*) verification results and cannot interpret the presence of uncertainty or inconsistency in the system's behaviors. Therefore, modelling and verifying these systems in the context of uncertainty or inconsistency raises the need for richer domains of truth values used to interpret the systems' behaviors and properties' satisfaction.

It is worth mentioning that multi-source data IoT and IS applications are highly susceptible to uncertainty and inconsistency because of their complexity, whether in their internal communications or the open environments in which they operate. This complexity that results from the existence of multiple sources of information makes modeling and verifying these kinds of systems a challenging task. Uncertainty occurs when there is missing information about a specific behaviour of the system under investigation, while inconsistency occurs when we have different viewpoints about a particular system's behaviour. For example, when two experts design the same system and disagree about some properties.

Therefore, our motivation in this research is to provide effective approaches for verifying MAS focusing on IoT and IS systems with commitment and trust protocols under uncertain or inconsistent settings where multiple data sources are involved [59] and multiple truth values can model the system. As argued in [49, 17, 58, 76], multi-valued model checking is a suitable technique for handling these types of models, but without involving the concepts of commitment and trust.

In this research, our starting point is intensively studying an efficient multi-valued model checking technique based on a multi-valued temporal logic that was initiated in [17]. The underlying logic of this technique, named mv-CTL is an extension of the two-valued CTL logic that generates satisfaction results beyond True or False. In particular, by using the multi-valued model checking, we can measure how close we are to true or false by considering multiple truth values mapped to a given lattice [84, 71]. Based on this, we generally planned to conduct the following scenarios: in reasoning about uncertainty over a multi-source data IoT/IS system with commitments and trust, the model checker should provide verification answers for commitment or trust formula among three values (T, M, F) based on the three-valued logic. The truth value M stands for *unknown* and is assigned to a given information that cannot be specified in the system under investigation. On the other hand, when reasoning about inconsistency, the model checker should give answers among (TT, FF, TF, FT) based on the four-valued logic, where TT means that there is an agreement between two parties about the satisfaction of a given property in a particular state of the system, FF denotes the agreement about the dissatisfaction and TF (resp. FT) means that the first party says *"Yes"* (resp. *"No"*), and the second party says *"No"* (resp. *"Yes"*) about the same property. These solutions lead to new and effective contributions to the field of system verification, where they overcome the shortcomings with the current related techniques explained in Table 1.1.

1.3 Problems and Research Questions

Our comprehensive analysis of commitment and trust within open systems has illuminated critical shortcomings stemming from the challenge of reasoning in the presence of incomplete or inconsistent information about system behaviors. Open systems, particularly within the realm of IoT/IS, entail a multitude of interacting components in dynamic environments, rendering them susceptible to uncertainties and inconsistencies in modelling their information. The prevailing methods for modeling commitments and trust consider only the absolute True or False for deciding the existence of system information. This limitation hinders their capacity to grapple with systems harbouring information with varying degrees of truth, thereby presenting a formidable verification obstacle. As a result, there arises an imperative for the development of efficient methodologies capable of autonomously assessing whether the behavior of IoT/IS systems aligns with their specifications.

So far, there is no approach for verifying IoT/IS systems with respect to certain properties related to commitments and trust under uncertain or inconsistent settings. Thus, our goal in this research is to deal with such an issue as a multi-valued model checking problems. In order to do so, our first step is to answer different important research questions.

Question 1. ***How can we define a temporal logic that is capable of specifying the commitments and trust properties from social prospective viewpoints with the presence of uncertainty or inconsistency?***

To address this question, we started by studying and investigating the possibility of using the existing commitment and trust logics [31, 27]. Our study revealed that these logics are incapable of modeling commitments and trust interactions of autonomous agents under uncertain or inconsistent settings. Therefore, we proposed to extend the existing logics of commitments called Computation Tree Logic for Conditional Commitments (CTL^{cc}) and Unconditional Commitment (CTL^{c}), and the logic of trust called Computation Tree Logic of Trust ($TCTL$) to the multi-valued versions. We mainly rely on the multi-valued logic introduced in [17]. Unlike the classical commitments and trust logics, in our produced logics, the modalities of commitments and trust are defined over multiple truth degrees based on given lattices, which give more expressive modelling language that enables us to express our models under uncertain and inconsistent settings.

Question 2. ***On which base do we make the new logics deal with multiple truth values for modelling and verifying IoT/IS systems?***

In formulating our new logics to accommodate multiple truth values for modeling and verifying IoT/IS systems, we anchored our approach in the foundations of lattice theory. This choice was instrumental in establishing the syntax and semantics of our logics and also in adeptly capturing the nuances of uncertainty and inconsistency inherent in these systems. By mapping the satisfaction degree of a formula onto a specific lattice, we gained a powerful framework for representing and evaluating uncertain and inconsistent information. This method yielded results characterized by precision, reliability, and a high degree of accuracy. This utilization of lattice theory forms a cornerstone of our methodology, providing a robust foundation for reasoning about complex IoT/IS systems.

Question 3. ***How can we formally verify the developed multi-valued temporal logics?***

In order to ensure that our approach increases confidence in reliability, safety and efficiency, we have put forward two main verification techniques:

- Direct algorithms that handle the multi-valued structure without reduction. The labelling of states and/or transitions is interpreted as elements from a lattice. After extensive study of these algorithms, we concluded that developing direct algorithms is harder than developing reduction ones. They require a lot of effort and time to be done, which makes it an inappropriate choice because of the time limitation assigned to finish this research.

- Reduction algorithms that reduce the multi-valued model checking problems to classical (two-valued) ones. The advantages of these algorithms are the ability to reuse efficient existing model checkers and their efficiency in dealing with the state explosion problem. In addition, they are not as complex as the direct ones. These algorithms enabled us to develop efficient and effective reduction-based multi-valued model checking techniques and opened a wide range of possibilities for verifying IoT/IS systems with different protocols within open environments.

Question 4 . ***How can we evaluate the proposed solution for the multi-valued model checking problem of the developed multi-valued logic?***

We applied two evaluation methods: (1) Empirical, which is evaluated by applying the proposed algorithms to multiple real-world case studies and reporting the experimental results, and (2) Theoretical, which handles the theoretical analysis by providing computational complexity analysis and soundness proofs for all our algorithms.

1.4 Methodology

The methodology we've outlined is encapsulated and substantiated by contributions in Table 1.2. At the beginning of this dissertation, we reviewed and evaluated relevant approaches that use computational CTL trust logic TCTL and commitment logic CTLC, including conditional and unconditional cases. In addition, we reviewed and evaluated the multi-valued case of CTL (mv-CTL). Our intensive review and evaluation of these logics showed that the existing approaches based on logics of trust and commitment are effectively used to capture, model and reason about trust and commitment protocols in multi-agent systems (MAS). However, these approaches cannot deal with systems designed with missing or inconsistent information where this kind of information is highly likely to exist in widespread MASs such as IoT and intelligent systems. The reason is that commitment and trust logics are based on the classical CTL logic, which uses only the absolute truth values True and False. To start solving this problem and find an appropriate solution, we looked for approaches based on logics that consider multiple truth values. We found the multi-valued case of CTL (mv-CTL) is effectively used for designing and reasoning about systems with uncertainty and inconsistency using multi-valued model checking technique. In Contribution ① , (Section 1.5), we applied this logic to a new domain and designed and verified an IoT system with missing information. We provided a mathematical representation and full implementation of the reduction-based multi-valued model checking algorithm.

However, since our focus is on open systems with uncertainty and inconsistency that involve trust and commitment protocols, mv-CTL doesn't include the modalities of trust and commitment, which makes it incapable of dealing with these kinds of critical systems. Therefore, we concluded that filling the gap between mv-CTL and the logics of trust and commitment is the appropriate way to have a complete logics that can be an efficient and effective solution to solve the problem. To do so, we extended the mv-CTL with the modalities of commitment [31] in Contributions ② and ③ , and introduced the new logic named multi-valued commitment logic (mv-CTLC), including its versions explained in the next section. This logic is effectively used for reasoning about uncertainty and inconsistency over IoT/IS systems with commitment protocols. Then, in Contributions ④ , we followed similar strategies to produce the multi-valued logic of trust mv-TCTL by extending the mv-CTL with the modalities of trust logic [24]. The new logic mv-TCTL proved its efficiency and practicability in verifying IoT/IS systems with trust protocols. Simultaneously with producing these logics, we developed reduction approaches that reduce the multi-valued logics to their classical case to take advantage of reusing the existing model-checking tools. We chose to apply the reduction approaches, not the direct ones, because of the latter's complexity and the long time and effort needed to develop them in a specific period determined for this research. Sequentially, to implement our approaches, we developed two tools that transform our logics to their classical cases and automatically interact with the well-known checkers NuSMV and MCMAS. These two tools are integrated and improved to be one tool called MV-Checker. The MV-Checker developed in Contribution ⑤ . It streamlines the process of verifying the properties of MAS, including IoT/IS systems, with trust and commitment within uncertain and inconsistent environments.

Finally, we modelled and verified multiple IoT/IS systems and conducted 12 verification experiments used to evaluate our approaches. We also compared the MV-Checker with another tool in the same field regarding performance and reliability.

1.5 Contributions

This thesis provides the following contributions:

Contribution ①

Our first contribution [2] focuses on conducting a practical experiment using the already exciting mv-CTL logic in dealing with verifying systems with uncertainty. In specific, we applied this logic to a new domain by modeling an IoT system called Smart Home with missing information in some states and specified a set of properties to be checked against this system. Moreover, we used a reduction algorithm to transform the 3v-CTL to CTL and verified our system over NuSMV checker. Our findings reported that the 3v-model checking approach based on 3v-CTL is efficient and effective in verifying these systems. However, this approach cannot deal with systems emphasizing trust and commitment protocols, which strongly exist in IoT/IS systems. We therefore planned to conduct Contribution ② .

Contribution ②

In the second contribution [3], we introduced our first three-valued unconditional commitment logic named 3v-CTL^c. Specifically, we introduced a scalable verification approach for IoT/IS applications in uncertainty-characterised settings with timed commitments using

Ste	Description	Input	Output and contribution
1	reviewing and evaluating	Current methods and evaluation criteria	• Limitations
2	Selecting a suitable logic	Ability to capture trust and commitment protocols/uncertainty and inconsistency	• CTL^c/CTL^{cc}, TCTL and mv-CTL
3	Applying 3v-CTL on a new domain/providing a new implementation algorithm	3v-CTL/theoretical algorithm	• Verifying IoT/IS systems with implementation • Contribution ①
4	Extending CTL^c with 3v-CTL^c	CTL^c with new modalities with their semantics rules	• 3v-CTL^c • Contribution ②
5	Extending CTL^{cc} with 3v-CTL^{cc}	CTL^{cc} with new modalities with their semantics rules	• 3v-CTL^{cc} • Contribution ②
5	Extending CTL^c with 4v-CTL^c	CTL^c with new modalities with their semantics rules	• 4v-CTL^c • Contribution ③
7	Extending CTL^{cc} with 4v-CTL^{cc}	CTL^{cc} with new modalities with their semantics rules	• 4v-CTL^{cc} • Contribution ③
8	Extending TCTL with 3v-TCTL	TCTL with new modalities with their semantics rules	• 3v-TCTL • Contribution ④
9	Extending TCTL with 4v-TCTL	TCTL with new modalities with their semantics rules	• 4v-TCTL • Contribution ④
10	Selecting approaches	Direct/Reduction	• Reduction • Contribution ③ and ⑤
11	Applying the new mv-logics on IS/IoT domains	mv-TCTL/mvCTLC	• Choosing systems, modeling and encoding • Contribution ② - ⑤
12	Implementation: Developing an mv-model checker	Coding/Testing/Verification	• MV-Checker • Verification, results, comparison and future work • Contribution ⑤

Table 1.2: Our methodology steps with contributions

three-valued model checking. This logic covers the commitment protocols that do not emphasize the conditional action needed to satisfy the desired action in the previous conditional commitment logic.

We used the new logic for reasoning about uncertainty in commitment protocols, modeled a smart contract-based IoT mortgage system with commitments under uncertain settings, introduced a set of specifications, produced an input formal language named 3v-ISPL+, and implemented a verification framework of our model against its specifications using a transformation algorithm and the MCMAS$^+$ model checker. Finally, we reported and discussed our experimental results, which proved the scalability and efficiency of our approach.

Later in [8], we started by proposing 3v-CTL^{cc}, a new modeling language that extrapolates the classical timed conditional commitment logic CTL^{cc} to the three-value case. We defined new semantics of the conditional commitment modality in this new logic. We also simulated and modeled a scenario of A Smart Hospital system and introduced a set of specifications. We verify the system model against these specifications using a reduction-based multi-valued model checking approach. The effectiveness of the proposed approach was evaluated by implementing it on the MCMAS-SC model checker. Next, we recognized the need for reasoning bout inconsistency over the systems under investigation as they are highly subject to inconsistency due to their rabid growth and complexity. Therefore, we presented Contribution ③ .

Contribution ③

In this work [7], we provided enhancements to our previous approaches and presented the following original contributions:

- Introducing two new logics, namely (4v-CTL^{c}) and (4v-CTL^{cc}) for modelling and reasoning about multi-agent and multi-source data systems with conditional and unconditional commitment protocols under inconsistent settings. Particularly, we apply these logics over IoT and IS, where the underlying multi-source data environments are subject to inconsistency due to the extensibility and complexity of their interactions. Moreover, we presented the corresponding input formal language 4v-ISPL+ and used a new model checking technique, four-valued model checking.

- Providing a formal representation (**Algorithm 1**) and full implementation of a preliminary reduction algorithm described by [14] that transforms the multi-valued CTL (mv-CTL) introduced in [17] to CTL.
- Producing four new reduction algorithms: **Algorithm 2**, **Algorithm 3**, **Algorithm 4**, and **Algorithm 5**. **Algorithm 2** transforms our recently produced three-valued conditional commitment logic (3v-CTL^{cc}) [8], and three-valued unconditional commitment logic (3v-CTL^{c}) [3] to CTL^{cc} and CTL^{c} respectively. **Algorithm 3** transforms the new 4v-CTL^{cc} and 4v-CTL^{c} to CTL^{cc} and CTL^{c}. **Algorithm 4** transforms our recently produced three-valued unconditional commitment logic (3v-CTL^{c}) [3] to CTL. While **Algorithm 5** transforms the new four-valued unconditional commitment logic 4v-CTL^{c} to CTL. The implementation of these algorithms allows us to reuse the existing model checking tools NuSMV and MCMAS+ to verify IoT and IS applications in multi-source data settings, which has not yet been attempted in the literature.
- Developing two Java-based tools for implementing our algorithms: **NuSMV-interactor** for the transformation to CTL and **MCMAS-interactor** for the transformation to the two-valued versions of the multi-valued logics.
- Performing multiple verification experiments to check the scalability of our approaches.
- Providing comparisons between the reduction approaches that reduce the multi-valued logics to CTL and the ones that reduce these logics to the two-valued versions. Based on these comparisons, we concluded that the reduction to CTL for using NuSMV checker is less scalable and takes more time than the reduction to the classical commitment logics and use MCMAS checker. Therefore, this opened a new future work direction, and we set a plan for improving NuSMV to handle commitment scenarios directly and produce a more efficient version of this checker.

As trust protocols are also important and widely used in IoT/IS and are subject to be designed with missing and inconsistent information, we decided in Contribution ④ to produce the three-valued and four-valued cases of the classical trust logic TCTL.

Contribution ④

In this contribution [5], we presented a new framework for verifying open systems with trust under uncertainty. Specifically, we addressed the problem using three-valued model checking. We introduced a new logic by extending the recently proposed Computation Tree Logic of Trust (TCTL) to the three-valued case(3v-TCTL) to reason about trust with uncertainty over smart contract-based systems. We also propose a new transformation approach to reduce the 3v-TCTL model checking problem to the classical case. Moreover, we presented a new formal input language 3v-VISPL. We apply our approach to a Smart Contract-based Drug Traceability system in the healthcare supply chain. The approach is implemented using a Java toolkit that automatically interacts with the NuSMV model checker. We verify this system against a set of specifications and report the results of our experiments.

Sequentially, in [4], we introduced a new framework for modelling and verifying IoT applications in the healthcare domain using four-valued model checking. We focused on applications that involve interactions based on trust protocols under inconsistency. Specifically, we introduced a new logic of trust called 4v-TCTL to reason about the inconsistency between designers over IoT systems. We used a Smart Glucose Monitoring System as a case study. We modeled our system and assigned a set of trust properties to be checked against this system. We introduced a new reduction algorithm for reducing our four-valued model checking problem to the two-valued version to reuse an existing model checker called MCMAS^t. Moreover, we presented a new formal input language 4v-VISPL. We verified our system using our approach and reported the experimental results.

Contribution ⑤

At this research stage, we focused on the implementation part and developed a general verification tool to transform all our new logics to CTL and their classical cases and interact with several model checkers [6].

- Our main motivation is developing a Java-based tool named **MV-Checker**, which streamlines the process of verifying the properties of MAS, including IoT/IS systems with trust and commitment within uncertain and inconsistent environments. The tool is a large extension to the tool developed in [23] and designed to perform the following functions: **(1)** Transforms our multi-valued logics of trust (3v-TCTL and 4v-TCTL) into TCTL and automatically interacts with the MCMSA^t model checker. **(2)** Transforms our multi-valued logics of commitment (3v-CTLC and 4v-CTLC) into CTLC and automatically interacts with the MCMSA+ model checker. **(3)** Additionally, the tool can transform these logics and mv-CTL logic into CTL and automatically interact with the NuSMV model checker. **(4)** Moreover, it allows directly verifying the classical CTL, CTLC and TCTL models. These functions make the **MV-Checker** an essential tool that efficiently and accurately verifies complex systems modeled in multi-valued logics of trust and commitment under uncertain and inconsistent environments.

- Technical contributions:
  - We develop two new reduction algorithms. The first algorithm transforms the 3v-TCTL logic to its classical counterpart, TCTL. The second algorithm transforms the 4v-TCTL logic to CTL. These new algorithms allow for more flexible and expressive representations of the systems under investigation and extensive comparisons between the algorithms that convert mv-TCTL to TCTL and the algorithms that transform this logic into CTL.
  - To check the effectiveness and efficiency of the proposed tool, we perform multiple verification experiments by modelling several IoT and intelligent applications. We include in this study two blockchain-based applications in the healthcare domain: Smart Drug Traceability and Smart Health Record Management systems. We model these applications with mv-TCTL of trust for reasoning about uncertainty and inconsistency and 4v-CTLC of commitment for reasoning about inconsistency.
  - We conduct various experiments to compare the approaches developed. The comparisons are between the results obtained from transforming the mv-TCTL to CTL and classical TCTL regarding scalability and accuracy. We also compared the performance of our tool with the performance of another tool already

developed in the same domain.

- We provide the packages of all our previous and recent case studies containing eleven experiments with SMV, ISPL+, VISPL, mv-ISPL+ and mv-VISPL files. Additionally, we provide the source code, the Jar file of the tool, and the user manual document explaining the use of the proposed tool.

1.6 Overview of the Dissertation

This section provides an overview of the dissertation and describes the content of its chapters. The chapters of this dissertation are classified into three main parts:
Part I: lays the groundwork, introducing and solidifying the core concept around which the entirety of the thesis revolves. It consists of Chapter 1 and Chapter 2.

- Chapter 1: introduces the context of the research and its motivations. It also gives an overview of our methodologies and contributions with a general overview of the dissertation.
- Chapter 2: provide a background which recalls the concepts and topics that are the basis for our research. These concepts are the base logics used to build our new logics, lattice theory, and model checking techniques.

Part II: is about our contributions and covers the following chapters:

- Chapter 3: covers our evaluations and testing of the existing multi-valued logic and its capability to deal with a new domain, which is verifying IoT systems.
- Chapter 4: introduces the first framework for modelling and verifying IoT/IS systems with our new logic of multi-valued commitment. It also covers our newly developed reduction algorithms with their soundness proofs and computational complexity analysis for each algorithm.
- Chapter 5: introduces the second framework for modelling and verifying IoT/IS systems with our new logic of multi-valued trust. It also covers the newly developed

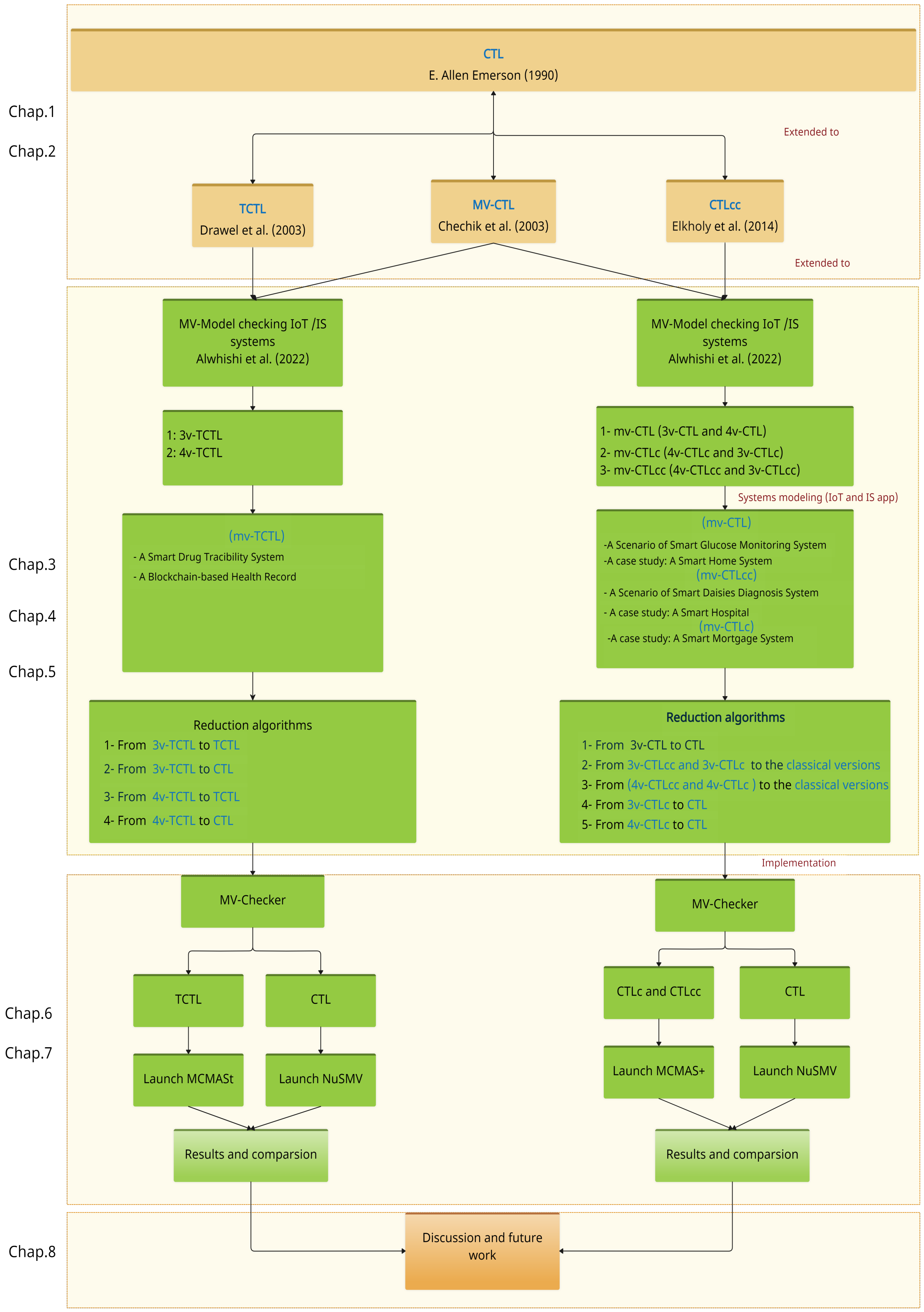


Figure 1.1: Overview of the dissertation

reduction algorithms with their soundness proofs and computational complexity analysis for each algorithm.

- Chapter 6: covers the implementation part and gives a deep picture of our new model checker. It explains the internal design of our tool and its interfaces.
- Chapter 7: covers our case studies and provides full modeling and verification of the systems on hand using the newly developed checker.

Part III: provides a deep analysis and discussions about our findings and the future work plan based on these discussions. Figure 1.1 gives an overview of the complete work done in this dissertation.

Chapter 2

Background

2.1 Computation Tree Logic (CTL)

Clarke and Emerson introduced Computation Tree Logic (CTL) in the early 1980s for systems specification and verification [67]. The idea behind CTL is that the underline modeling time in this logic is assumed to have a tree-like structure where each moment in time can be divided into different possible paths in the future.

2.1.1 CTL model

The model of CTL is a system model represented by a Kripke structure, which is a finite directed graph consisting of nodes representing states and edges that represent transitions and are used for describing systems' behaviors. Formally, Kripke structure is a tuple $D = (AP, S, S_0, LF, R)$ where AP is a finite set of atomic propositions, S is a finite set of states, $S_0 \subseteq S$ is a set of initial states, $LF : S \to 2^{AP}$ is a labelling function that maps a set of propositional variables to states where these variables hold. $R \subseteq S \times S$ is a transition relation between states.

2.1.2 Syntax

Let AP be a set of atomic propositions, then:

- $\top, \bot$ and every atomic proposition $a \in$ AP are formulae;

- If ϕ and ψ are formulae then so are $\neg\phi$, $\phi \wedge \psi$, $\phi \vee \psi$, EXϕ, AXϕ, EFϕ, AFϕ, E[$\phi \cup \psi$], A[$\phi \cup \psi$], EGϕ, AGϕ.

  where A and E are universal and existential path quantifiers. The state quantifier operators are F, which represents "sometime in the future", X means "in the next state", $\neg$ represents the negation, G means globally along the system path, and $\cup$ means "until".

2.1.3 Semantics

Let a system model represented as a Kripke structure $D = (AP, S, S_0, L, R)$. Let a satisfaction relation $(D, s) \models \phi$, which means the formula ϕ holds at state s in model K. The symbols $\models$ and $\not\models$ represent satisfaction and dissatisfaction, respectively. Let ϕ_1, ϕ_2, ψ be formulae then:

- $(D, s_0) \models a$ iff $a \in L(s_0)$ means state s_0 satisfies an atomic proposition a if a is in the labelling set of state s_0;
- $(D, s_0) \models \neg\phi_1$ iff $(D, s_0) \not\models \phi_1$;
- $(D, s_0) \models \phi_1 \vee \phi_2$ iff $(D, s_0) \models \phi_1$ or $(D, s_0) \models \phi_2$;
- $(D, s_0) \models \phi_1 \wedge \phi_2$ iff $(D, s_0) \models \phi_1$ and $(D, s_0) \models \phi_2$;
- $(D, s_0) \models \mathbf{EX}\psi$ iff $\exists$ state t: $(s_0, t) \in R$ and $(D, t) \models \psi$;
- $(D, s_0) \models \mathbf{AX}\psi$ iff $\forall$ state t, if $(s_0, t) \in R$ then, $(D, t) \models \psi$;
- $(D, s_0) \models \mathbf{A}[\phi_1 \cup \phi_2]$ iff for **every** path $s_0, s_1, \ldots \exists i \geq 0$: $(D, s_i) \models \phi_2$ and $\forall$ $0 \leq j < i$, $(D, s_j) \models \phi_1$;
- $(D, s_0) \models \mathbf{E}[\phi_1 \cup \phi_2]$ iff for **some** path $s_0, s_1, \ldots \exists i \geq 0$: $(D, s_i) \models \phi_2$ and $\forall$ $0 \leq j < i$, $(D, s_j) \models \phi_1$;
- $(D, s_0) \models \mathbf{EG}\phi$ iff there exists a path $\pi = s_0, s_1, \ldots$ in D starting from s_0 such that $\forall i \geq 0$, $\pi(i) \models \phi$;

- $(D, s_0) \models \mathbf{AG}\phi$ iff for all paths $\pi = s_0, s_1, \ldots$ in D starting from s_0, we have $\pi(i) \models \phi$ for all $i \geq 0$

2.2 Computational Logic of Social Conditional Commitments (CTL^{cc})

The concept of social commitments effectively applies to classical (no missing or inconsistent information) IoT/IS models where their intelligent agents commit to performing a particular action towards another agent when the latter fulfils a particular condition. In this section, we recall the conditional commitment logic produced in [31] and used for capturing commitment protocols within multi-agent systems. For example, In the context of conditional commitment logic, in a smart home scenario, this logic enables the capture of a property that states, *"When the user sets a specific washing time, the smart washing machine commits to washing the clothes within this specified time"* . The condition in this scenario is *" specifying the time by the user"* to make the smart machine commit to washing clothes within this time. Below, we define an IoT-based system with conditional commitment protocols.

2.2.1 CTL^{cc} model

IoT Model of CTL^{cc}: an IoT model with timed conditional commitments is a tuple $K = (S, R, \{\sim_{i\to j} | \ (i,j) \in A^2\}, I, V)$ where:

- A is a set of system agents;
- S is a set of global states of the IoT system;
- $R \subseteq S \times S$ is a total transition relation for the IoT dynamics;
- $I \subseteq S$ is a set of initial global states;
- V:AP$\rightarrow 2^S$ is a valuation function that maps every atomic proposition $a \in$ AP to a set of states;

- For each pair of agents $(i, j) \in A^2, \sim_{i \to j} \subseteq S \times S$ is a social accessibility relation defined by $s \sim_{i \to j} s^{'}$ iff:

  (1) $l_i(s) = l_i(s^{'})$, where $l_i(s)$ and $l_i(s^{'})$ represent the local states of agent i in the global states s and $s^{'}$, where a global state signifies the instantaneous configuration or current status of all agents within the multi-agent system at a specific moment;

  (2) $(s, s^{'}) \in R$ is a transition relation;

  (3) $Var_i \cap Var_j \neq \emptyset$ and $\forall x \in Var_i \cap Var_j$ we have $l_i^x(s) = l_j^x(s^{'})$, where Var_i and Var_j are local variables and x is the shared variable that represents the communication channel between i and j; and

  (4) $\forall y \in Var_j - Var_i$ we have $l_j^y(s) = l_j^y(s^{'})$ where y is the variable of agent j and unshared with agent i.

In summary, the commitment accessibility relation is introduced to indicate in which states of the system the commitment takes place among the system's agents. The shared variable is used as a tag to move from the current state to a state where the commitment holds. Every communication channel is indicated by one shared variable between two agents. For example, the communication between agents i and j is captured by one shared variable, $x = 1$, for instance. If there is another communication between i and another agent z, then another shared variable should be assigned between these two agents.

2.2.2 Syntax

The syntax of CTL^{cc} is defined as follows:

$\phi ::= p \mid \neg\phi \mid \phi \vee \phi \mid EG\phi \mid EX\phi \mid E(\phi \cup \phi) \mid CC,$

$CC ::= CC_{i \to j}(\phi, \phi)$

where $\neg, E, A, \cup$ are as defined in Definition 2.1.2 and i and $j \in A$ are two agents; CC represent the conditional commitments.

The syntax of the formula $CC_{i \to j}(\psi, \phi)$ means "agent i commits towards agent j to bring about ϕ if the condition ψ holds".

2.2.3 Semantics

The semantics of this logic extend the semantics of CTL by adding the commitment modality $CC_{i \to j}(\psi, \varphi)$. The satisfaction relation represented by $(K, s) \models CC_{i \to j}(\psi, \varphi)$ where K is an IoT commitment model, s is a global state and $CC_{i \to j}(\psi, \varphi)$ is a conditional commitment formula is defined as follows:

- $(K, s) \models CC_{i \to j}(\psi, \varphi)$ *iff* (1) $\exists s^{'} \in S$ *s.t.* $s \sim_{i \to j} s^{'}$ and $(K, s^{'}) \models \psi$ *and* (2) $\forall s^{'} \in S$ *s.t.* $s \sim_{i \to j} s^{'}$ *and* $(K, s^{'}) \models \psi$, we have $(K, s^{'}) \models \varphi$.

This semantic means the conditional commitment formula "agent i commits to bring about φ for responding to agent j after fulfilling the condition ψ" is satisfied in state s of the IoT commitment model K if and only if there is at least one accessible state $s^{'}$ where ψ holds and whenever the latter holds, φ as well holds in the same state.

Example 1. To explain the idea of the commitment accessibility relation and the semantics of this logic, consider the system model in Figure 2.1. In this model, according to the semantics, state s_0 satisfies the given commitment formula where the following two conditions hold:

- There is at least one accessible state, s_1, accessible from s_0. This commitment accessibility relation is represented by the shared variable x, which has the same value (x=1) associated with the two variables Var_i in s_0 and Var_j in s_1.
- In this accessible state, the condition of *"whenever ψ holds in an accessible states, φ holds as well in the same state"* is satisfied in s_1.

2.3 Computational Logic of Social Unconditional Commitments (CTL^c)

The model of this logic is defined as the model of the conditional commitment logic. Bellow we explain the syntax and semantics of this logic

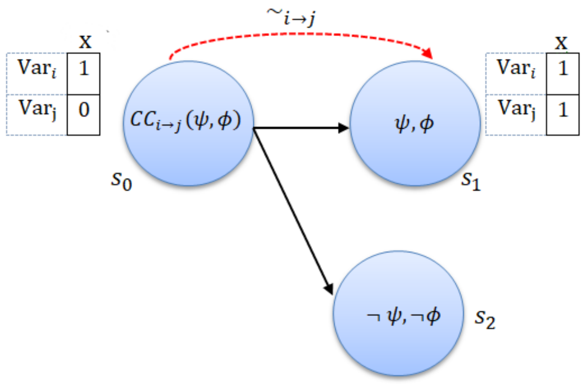


Figure 2.1: An example of the CTL^{cc} model

2.3.1 Syntax

The syntax of CTL^c is similar to CTL^{cc} with one difference in the commitment formula where this formula is denoted by $C ::= C_{i\rightarrow j}(\varphi)$. Meaning that agent i commits towards agent j to bring about φ.

2.3.2 Semantics

Let U is a CTL^c model, then $(U,s) \models C_{i\rightarrow j}(\varphi)$ *iff* $\forall s^{'} \in S$ *s.t.* $s \sim_{i\rightarrow j} s^{'}$ we have $(U,s^{'}) \models \varphi$. This semantics means state s of the IoT/IS commitment model U satisfies the commitment formula "agent i commits towards agent j to bring about φ " if and only if all the accessible states $s^{'}$ hold the formula φ.

2.4 Computational Logics of Trust (TCTL)

This logic expands the Computation Tree Logic (CTL) established by Clarke and Emerson in the 1980s [67] to TCTL by adding trust modalities to capture the trust protocols among agents over global states of the system.

2.4.1 TCTL model

The model of TCTL is a tuple $(\mathcal{D}) = (S, R, \{\sim_{i\rightarrow j} | \ (i,j) \in A^2\}, \text{I}, Fn)$ where:

- A is a set of system agents;

- S is a set of reachable global states;
- $R \subseteq S \times S$ is a transition relation that covers all possible states;
- $I \subseteq S$ represents a collection of initial global states;
- $Fn : S \to 2^A$ is a valuation function relates the atomic propositions in A to a given state;
- For each pair agents $(i, j) \in A^2, \sim_{i\to j} \subseteq S \times S$ represents the vector-based trust accessibility relation denoted by $s \sim_{i\to j} s^{'}$ and defined in [29]. These accessibility relations define states that are compatible respectively with the trust vision fulfilment vision of the agents with respect to each other.

2.4.2 Syntax

: $\phi ::= a \mid \neg\psi \mid \psi \vee \psi \mid EG\psi \mid EX\psi \mid E(\phi U\psi) \mid T(i, j, \phi)$

where a is an atomic proposition and, ψ and ψ are formulae. The syntax $EG\psi$ means "a path exists in the system where the formula ψ holds in all the states". The formula $T(i, j, \phi)$ represents "agent i trusts agent j to bring about formula ϕ .

2.4.3 Semantics:

The semantics of this logic extend the CTL's semantics by adding the trust modalities. Given that, $\mathcal{D}$ is a trust model, s is a global state, and ψ is a trust formula, The satisfaction relation represented by $(\mathcal{D}, s) \models \psi$ is defined as: $(\mathcal{D}, s) \models T(i, j, \phi)$ *iff* $s \not\models \phi$ *and* $\forall s^{'} \neq s$ such that $s \sim_{i\to j} s^{'}$, we have $(\mathcal{D}, s^{'}) \models \phi$. This semantic means the state s in the trust model $\mathcal{D}$ satisfies the trust formula that says "agent i trusts agent j to bring about formula ϕ" if the current state doesn't satisfy formula ϕ and this formula is satisfied in all the trusted, accessible states that cannot be the current state.

2.5 Lattice Theory

It is known that reasoning based on certain information depends on two-valued, two-valued logic. The need to reason about uncertain or inconsistent information is increasing with the rapid growth of IoT and IS components and their complex communications. Therefore, we need a logic that considers several levels of truth degrees that are decided depending on the application on hand. This new logic has truth values over a lattice [19, 60] and produces one kind of many-valued logic called lattice–valued logic [84, 71].

A lattice is a structure $(L, \sqcup, \sqcap)$. Every two elements x and y in this structure have a join denoted by $(x \sqcup y)$ and a meet denoted by $(x \sqcap y)$. This structure satisfies important laws such as identity, commutativity and distributivity.

The concept of the lattice is used to establish the lattice-valued logic, which is commonly referred to as multi-valued logic. Below, we illustrate various types of lattices and their association with multi-valued logics.

- A two-valued logic, which is also called the classical logic, is given in the lattice in Figure 2.2 (a), with ¬T = F and ¬F = T. This logic used for modelling systems when the atomic propositions in states take truth values with absolute true or false.
- The lattice in Figure 2.2 (b) gives three-valued logic. T stands for *true*, F stands for *false*, and M stands for *maybe*, where ¬T = F, ¬F = T, ¬M = M. This logic is known as Kleene's strong 3-valued logic [51]. It is effectively employed for reasoning about abstracted or partial models [18] and serves as an effective technique to address state explosion problems. In terms of abstraction, this is achieved by approximating sets of realistic states with abstract states and/or approximating sets of realistic transitions with abstract transitions. The lattice has two join-irreducible elements T and M where. These elements are used in the reduction algorithm, as explained in the coming chapters.
- The lattice in Figure 2.2 (c) gives Belnap's 4-valued logic, T: *true*, F: *false*, N: neither true nor false, and B: both true and false, where ¬N = N and ¬B = B. This logic can

be used to reason about inconsistent information stored in a computer database [11].

- The lattice in Figure 2.2 (d) gives the product algebra 2×2 and takes truth values TT, FF, TF and FT where ¬TF = FT and ¬FT = TF. This logic is effectively employed for reasons about disagreements between two knowledge sources[17]. The meaning of the truth values of this logic can be understood by taking an example of two opposing parties where TF indicates that the first party says that the specification is true (satisfied) while the second says it is false (not satisfied).

The join-irreducible elements of the lattice in (c) and (d) are (B,N) and (TF,FT), respectively.

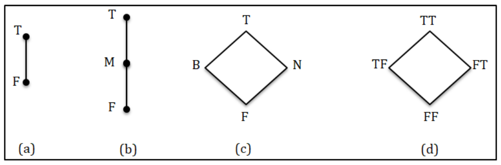


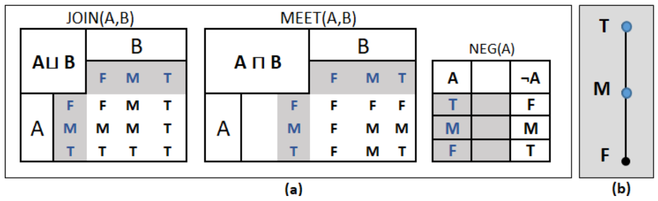
Figure 2.2: Types of lattices

2.5.1 Three-Valued Lattice Logic (3v-Logic)

This logic relies on the implication of the 3v-valued lattice (L_3). This lattice has truth values similar to Kleene's logic [84, 71], which has three truth values (T, M, F). T mans True, M to present the uncertain or messing information, and F means False. Figure 2.3 (b) shows the three-valued lattice, while Figure 2.3 (a) shows the truth values of this lattice. The operators $\sqcup$ and $\sqcap$ work as (AND) and (OR) operators in Kleene's logic.

2.5.2 Four-Valued Lattice Logic (4v-Logic)

This logic is based on the four-valued lattice shown in Figure 2.4 (b) with truth values taken from the product lattice 2×2 where these values are TT, FF, TF and FT. This logic can be used to reason about disagreements between two knowledge sources or conflicting viewpoints [17, 44]. Figure 2.4 (a) shows the truth values of this lattice.



JOIN(A,B)

A⊔ B		F	M	T
A	F	F	M	T
	M	M	M	T
	T	T	T	T

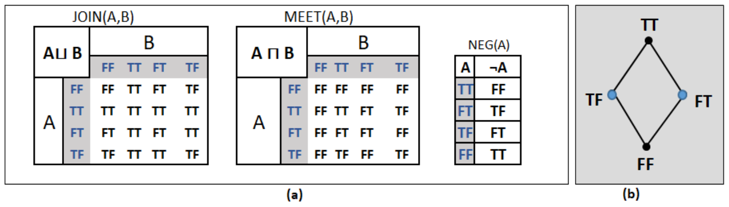
MEET(A,B)

A ⊓ B		F	M	T
A	F	F	F	F
	M	F	M	M
	T	F	M	T

NEG(A)

A		¬A
T		F
M		M
F		T

Figure 2.3: (a) The truth table of three-valued lattice; (b) Three-valued lattice L_3



JOIN(A,B)

A⊔ B		FF	TT	FT	TF
A	FF	FF	TT	FT	TF
	TT	TT	TT	TT	TT
	FT	FT	TT	FT	TT
	TF	TF	TT	TT	TF

MEET(A,B)

A ⊓ B		FF	TT	FT	TF
A	FF	FF	FF	FF	FF
	TT	FF	TT	FT	TF
	FT	FF	FT	FT	FF
	TF	FF	TF	FF	TF

NEG(A)

A	¬A
TT	FF
FT	TF
TF	FT
FF	TT

(a)
TT
TF
FT
FF
(b)

Figure 2.4: (a) The truth values taken from the product algebra 2×2; (b) Four-valued lattice

2.5.3 Join-Irreducible Elements

Definition 1. *A join-irreducible element in lattice L is an element $z \in L$ where $z \neq \perp$ and for any $x, y \in L$, if $z = x \sqcup y$, then either $z = x$ or $z = y$. $\perp$, in our case, represents the value FF. Let L be a partial order $(L, \leq)$. An element $x \in L$ is called a join-irreducible element iff $x \neq \perp$ and , for any $a, b \in L$, if $x = a \sqcup b$, then either $x = a$ or $x = y$. The set of the join-irreducible elements of a given lattice is denoted by $\mathcal{JI}(\mathcal{L})$.*

Intuitively, the Join-irreducible elements cannot be $\perp$ or decomposed into two other elements. In lattice L_3, the Join-irreducible elements are T and M; in lattice L_4 the Join-irreducible elements are TF and FT with color blue in Figure 2.4, (b). Every element $a \neq \perp$ of a finite distributive lattice can be uniquely decomposed into the join of all join-irreducible elements in its downward closure [19]. Formally, $a = \bigcup(\mathcal{JI}(\mathcal{L}) \cap \downarrow a)$. These elements, with their useful properties, will be the base of our reduction approaches from the multi-valued to the two-valued model checking problems.

2.6 Multi-Valued CTL (mv-CTL)

Multi-valued CTL denoted by mv-CTL is an extension of CTL where the atomic propositions and/or transitions between states take truth values over a lattice [17]. Our work focuses on the three-valued propositional logic based on the three-valued lattice ($3v{-}lattice$) and four-valued propositional logic based on the four-valued lattice ($4v - lattice$) that are explained in the previous section.

2.6.1 mv-CTL model

The mv-CTL IoT/IS model is a tuple $\mathbb{D} = (AP, S, \mathbb{R}, I, \mathbb{O})$ where:

- AP is a set of atomic propositions;
- S is a set of global states for the system;
- $\mathbb{R} \subseteq S \times S$ is a total transition relation denoted by $(s, s^{'}) \in R$ and takes truth values over lattice L;
- $I \subseteq S$ is a set of initial global states;
- $\mathbb{O} : S \to (AP \to L)$ is a total labeling function that maps every atomic proposition $x \in AP$ in $s \in S$ to a value in lattice L. Lattice L here can be L_3 or L_4 depending on the logic used.

Figure 2.5 shows an example of an mv-CTL model mapped to a three-valued lattice. The model is designed under uncertainty or missing information in states and in the transition between states where some formulae (atomic propositions) and transitions take truth values M, which captures the presence of missing information.

2.6.2 Syntax

The syntax is the same as the CTL syntax, except formulae are evaluated over a given lattice.

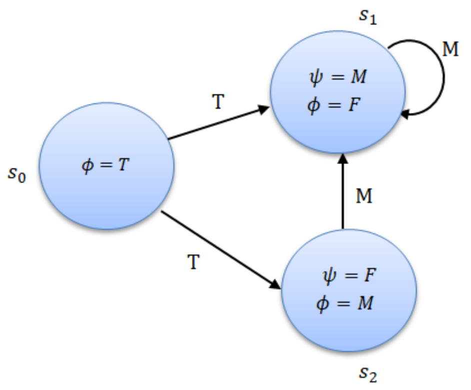


Figure 2.5: An example of an mv-CTL model mapped to a three-valued lattice

2.6.3 Semantics

The semantics of this logic extends the semantics of CTL as follows. In this semantic, the formulae are between double lines to indicate that we no longer consider the absolute true or false, but we consider how close we are to true or false.

Given a mv-CTL IoT/IS model $\mathbb{D}$ and a formula, the truth degree of the satisfaction of this formula is defined as follows:

- $\parallel a \parallel (s) = (\mathbb{O}(s))(a)$ where $a \in AP$ and $\mathbb{O} : S \to (AP \to L)$ is a total labeling function that maps states in S into L on a set of atomic propositions AP;

- $\parallel \varphi \vee \psi \parallel (s) = \parallel \varphi \parallel (s) \sqcup \parallel \psi \parallel (s)$ means in which truth degree, φ or ψ holds in state s;

- $\parallel \varphi \wedge \psi \parallel (s) = \parallel \varphi \parallel (s) \sqcap \parallel \psi \parallel (s)$ means in which truth degree, φ and ψ holds in state s;

- $\parallel \neg\varphi \parallel (s) = \overline{\parallel \varphi \parallel}(s)$ means in which truth degree, φ doesn't hold in state s;

- $\|EX\varphi\| (s) = \ pre^R_\exists(\| \varphi \|)(s) = \bigsqcup_{t \in S} \Big(\parallel \varphi \parallel(\text{t}) \sqcap \mathbb{R}(\text{s,t}) \Big)$ where $pre^R_\exists(\| \varphi \|)(s)$ denotes the backward image of state s that determines the value of φ in the next state. The semantics means in which truth degree there exists a path in the system where φ holds in the next state;

- $\| AX\varphi \| \ (s) = pre^{R}_{\forall}(\| \varphi \|)(s) = \prod_{t \in S} \Big(\| \varphi \| (\mathrm{t}) \sqcup \neg \mathbb{R}(\mathrm{s,t}) \Big)$ where $pre^{R}_{\forall}(\| \ \varphi \ \|)(s)$ denotes the backward image of state s that determines the value of φ in the next state of all paths;

- $\| EG\varphi \ \| = \nu\mathbb{Z}. \ \| \ \varphi \ \| \ \cap_L \ \| EX\mathbb{Z} \ \|$where $\nu\mathbb{Z}$ stands for the greatest fixed point of the globally operator G. The semantics expresses in which truth degree there exists a path in the system where φ globally holds;

- $\| E[\varphi \cup \psi] \| = \mu\mathbb{Z}. \ \| \ \psi \ \| \ \cup_L (\| \ \varphi \ \| \ \cap_L \ \| EX\mathbb{Z} \ \|)$ where $\mu\mathbb{Z}$ represents the smallest fix point of the formula $\varphi \cup \psi$. The semantics define in which truth degree there is a path where φ holds until ψ holds using the smallest fix point of $\varphi \cup \psi$.

Why multi-valued logic for IoT and IS in multi-source data settings?
IoT and IS in multi-source data environments are considered expanded systems because of their rapid growth and the large number of communicating and interacting intelligent agents within these systems in open settings. This makes the systems vulnerable to uncertainty and inconsistency regarding the system's interactions or the methods used to design these systems. Inconsistencies can occur during the design phase when system designers have conflicting views about some of the system's behaviors. This could stem from multiple reasons:

- Ambiguity: The system requirements or specifications may be ambiguous or open to interpretation. Different designers may interpret the requirements differently, leading to different views on the system's behaviors.

- Subjectivity: Designers may have different subjective opinions, perspectives, or preferences that influence their understanding of the system's behaviors. This can result in varying viewpoints and interpretations.

- Expertise and background: Designers may come from diverse backgrounds and have different levels of expertise or experience. Their knowledge, skills, and past experiences can shape their understanding of the system, leading to different viewpoints.

- Design constraints: Designers may face various constraints, such as time, budget, or technological limitations. These constraints can influence their design decisions and perspectives on the system's behaviors.
- Stakeholder perspectives: Different stakeholders, such as clients, end-users, or regulatory bodies, may have conflicting requirements or expectations for the system. Designers may need to consider these perspectives, which can lead to different views on the system's behaviors.

On the other hand, uncertainty can occur for the following reasons:

- Partial state space of IoT/IS models - some behaviours are unknown [13, 14].
- IoT/IS systems abstraction techniques to reduce the system state space [17].
- Open IoT/IS systems - external components determine some behaviours.
- Uncertain IoT/IS systems - some behaviors report conflicting information.

2.7 Model Checking

Model-checking [43] is a formal automated verification method used to assess the correctness of both hardware and software systems. Initially developed in the 1980s by Allen Emerson and Clarke, as well as by Sifakis and Queille, it plays a crucial role in ensuring system reliability. Model checking techniques can be classified into two groups: classical and multi-valued model checking. These groups are explained below.

2.7.1 Classical Model Checking

Classical model checking systematically examines a model's possible states or paths to verify if it adheres to a desired specification. Model checking is widely applied in hardware and software verification to ensure the correctness and reliability of systems. The output of the model checker can be one of three possibilities: a) Property Satisfied (T). b) Property Unsatisfied (F), accompanied by the generation of a counterexample that elucidates the cause of the satisfaction failure. c) State Explosion, which results from the exponential growth of the system state space due to an increase in state variables. The process of this method can be succinctly represented by the formula: $M \models \phi$ where M is a model and ϕ is a given formula. System specifications typically comprise propositional connectives $(\vee, \wedge, \neg, \rightarrow)$, temporal connectives, and primitive properties of individual states.

Various types of logic are employed in model checking techniques, contingent on the specific model checking tool and the application in question. These logics are categorized based on how they handle time. As such, logical formulas may be written in linear temporal logic (LTL), computation tree logic (CTL), μ-calculus, or monitor automata.

A range of model checking tools has been developed, each with distinctive features such as programming, modeling, and property languages, as well as graphical user interfaces (GUIs). Some notable tools include SPIN, UPPAAL, and NuSMV.

2.7.2 Multi-Valued Model Checking

Model-checking, in some cases, is insufficient to verify models with inconsistency, incomplete information or behaviors expressed in multiple truth values. Therefore, multi-valued model checking methods with many values logic are generated from classical model checking to deal with the following cases [44]:

- For reasoning about very large (or even infinite) state spaces [13].
- Disagreement and inconsistencies. This occurs during the software engineering process, where stakeholders may disagree about how the systems should behave during the design and implementation phase [30, 17].
- Uncertainty, when we have incomplete information about the real system, usually during the requirements analysis phase. Uncertainty occurs after removing some information during system abstraction or in case of an incomplete understanding of system properties.
- Relative importance: when we want to classify some behaviors as essential and others may or may not be considered to be implemented [17].

- Temporal logic query checking: to discover properties of systems. For example, a query on a system of the form $AX?\varphi$, means what is the property that holds on all paths and at the next states starting from a state s?

  In this method, the system model and its specifications take truth values over a lattice, and we no longer consider the truth values as True or False; instead, we measure how close to being true or False [15]. Deciding how many values of logic to consider depends on how we wish to combine information from individual viewpoints. The ranking on the elements of the lattice can be interpreted according to the application on hand, for example, in case we want to preserve information about which party said True and which party said False; in case we want to allow viewpoints to say ”don't know” for some propositions as in three-valued logic. In this approach, the temporal logic formula is interpreted on a multi-valued system model where the atomic proposition in every state is interpreted as an element in a lattice [44].

Generally, multi-valued model checking algorithms can be classified into two groups:

- Reduction algorithms that reduce the multi-valued model checking problems to classical ones. Using these algorithms sometimes leads to losing some information in the system under investigation. However, the advantage of these algorithms is the ability to reuse existing model checkers for verifying IoT system models [52, 46, 50, 8, 5, 3, 2]
- Direct algorithms that handle the multi-valued structure without reduction. The labelling of states and/or transitions is interpreted as elements from a lattice. Developing direct algorithms is harder than developing reduction ones. However, direct algorithms are more general and can adapt to various scenarios and settings [17, 58, 77].

Chapter 3

Applying mv-CTL to IoT Domain

3.1 Motivation and Contribution

The surge in the adoption of IoT (Internet of Things) technology [74] has led to the development of expansive systems interconnecting numerous smart components. These components interact within a network to execute various functions. However, uncertain and inconsistent information becomes pervasive within this intricate web of connectivity. As the number of interconnected devices grows, so does the prevalence and impact of these factors, amplifying the complexity and challenges inherent in managing and ensuring the reliability of these systems[79]. Using classical model checking in Verifying large IoT systems under uncertain or inconsistent settings is inefficient . The reason is that, as we mentioned earlier, we need a logic with additional truth value to capture the presence of uncertainty in a system. Therefore, we prove in this chapter that using the three-valued case of mv-CTL, 3v-CTL, is the appropriate solution. This logic is based on the three-valued lattice (L_3) with truth values (T, M, F). The value M is used to represent uncertain information in the system. We also show that 4v-CTL effectively applies to model and verifying such systems with inconsistency between system designers using the truth values (TF, FT, TT, FF).

This chapter explains the mv-CTL logic and applies it to a new domain, IoT systems. Particularly, we contribute by applying a practical reduction-based 3-valued model checking technique to verify IoT under uncertain settings. 3v-valued model checking is a particular case of multi-valued model checking proposed in [17]. We provide, for the first time, a mathematical and practical representation of the reduction algorithm produced in [13]. Moreover, we provide a full implementation of this algorithm.

3.2 Modeling Uncertainty in IoT Systems with 3v-CTL

This logic is efficient for reasoning about the uncertainty that stems from:

- Partial IoT models - some behaviours are unknown. This occurs when we have a big open system that needs to be verified, and we miss some parts or behaviors of this system.
- Abstracted IoT systems - some behaviours are disregarded to reduce the state space. This occurs when we abstract the system states and relations to overcome the state explosion problem.
- Open IoT systems - some behaviours are defined by other components or features.
- Uncertain IoT models - some behaviors are conflicting or reporting conflicting information at a specific phase of the IoT system life cycle or missing information about the behaviors of a given system.

Example 2. Consider a 3-valued model of an IoT system scenario where the formulae encoding the requirements take truth values over the lattice L_3 (see Figure 3.1). The variables a and b represent a specific position of an IoT component, for example, a smart coffee machine and a cup, respectively. $a = F$ in s_1 stands for: “it is false that the machine is ready” and $a = M$ in s_0 stands for “it is unknown whether the machine is ready or not”. $b = T$ in the same state stands for “the cup is ready”. The transition from s_2 to s_1 takes the value M because “it is unknown whether this transition exists or not in the system”.

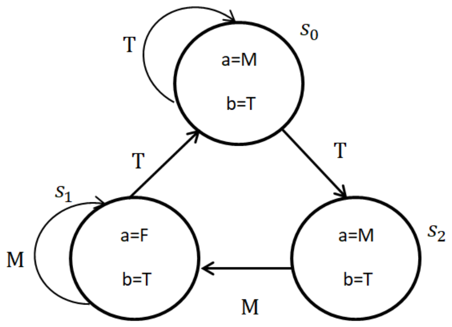


Figure 3.1: A scenario of a 3-valued smart coffee machine model

3.3 Modeling Inconsistency in IoT Systems with 4v-CTL

Below, we describe how we model our systems using 4v-CTL and how this logic relies on the multi-valued sets and relations to compute satisfactory results.

Example 3. Modeling a scenario of Smart Glucose Monitoring system with 4v-CTL

Consider the 4v-CTL model (Definition 2.6.1) of a Smart Glucose Monitoring system (denoted here by $\mathbb{A}$) depicted in Figure 3.2, which describes a specific scenario that shows how data transform from wearable devices used by a diabetic for monitoring the levels of glucose and temperature in his/her body. The truth values (TT, FT, FF, TF) are assigned to the atomic propositions as specified by the function $\mathbb{O}$ in the definition. We assume two designers designed the system based on different points of view and have disagreements about some of the system's behaviors. The system contains four states with variables that take truth values mapped to the four-valued lattice. The system starts when the glucose and temperature sensors read and send data to a related smart application installed on the patient's smartphone. In s_0, the variable "PatCon: TT" means there is an agreement (both say true) between the designers in this state about *the patient being connected to the sensors and the smartphone*. In s_1, the variable "InfoSend: TT" represents the agreement about *the sensor sends the read data to the smartphone* and "Alarm: FT" represents the disagreement (the first designer says false and the second says true) about *the sensor sends an alarm to the patient's smartphone when the first reads a high glucose level.* The variable "InfoSned: TT" in s_1 represents the agreement about *the temperature sensor sends data* *to the patient's smartphone.* The same applies to "SmPHRecive: TT" in s_3 where this variable stands for *the patient's smartphone receives the data read by the sensors.* The variable "SendAppReq: TF" represents the disagreement (the first designer says true and the second says false) about *the application sends an approval request to the user before the collected data is shared with the hospital.* For simplicity, we omitted the values FF representing the agreement (both designers say no) about the variables and transitions. Since this work focuses on the uncertain or inconsistent information in the states (atomic propositions), we assume the relations between states are true or false in our systems. Therefore, all the relations between the states in this system are mapped to TT, and to simplify, we neglected the relations with values FF.

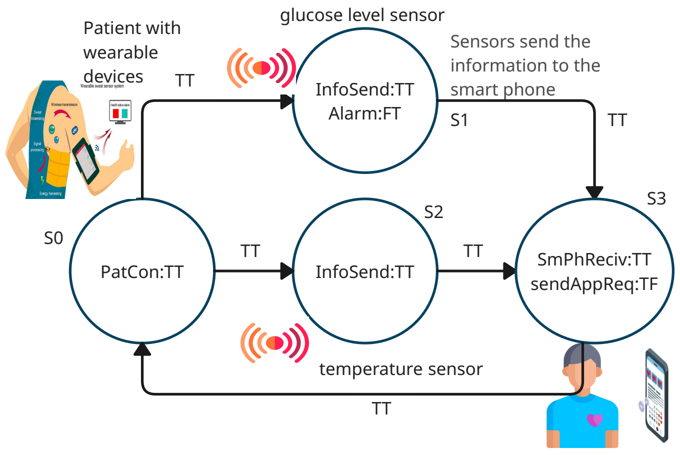


Figure 3.2: 4v-CTL model of a Smart Glucose Monitoring system ($\mathbb{A}$)

A. Four-Valued Sets (4v-sets)

A 4v-set is a function ($\mathbb{O} : S \to L_4$) where S is a set of elements and L_4 is a four-valued lattice. Thus, $\mathbb{O}(a)$ stands for the degree of membership of a in the function $\mathbb{O}$. The 4v-sets of the variables *InfoSend* and *Alarm* represented by $\|$InfoSend $\|$ and $\|$ Alarm $\|$ respectively and the multi-valued sets of the meet of these variables ($\|$ InfoSend $\| \sqcap \|$ Alarm$\|$) are shown in Figure 3.3. In (a), ‖InfoSend ‖ is obtained as follows: s_1 and s_2 are mapped to the truth value TT in the four-valued lattice because the variable "InfoSend" has the value TT in these states. The same variable doesn't have the value TF or FT in the system; therefore, no state is mapped to these values in the lattice. While it has the value FF in states s_0 and s_3, these states are mapped to the value FF in the lattice. The same applies to the mv-set of variable "Alaram" (‖ *Alarm* ‖) in (b) where state s_1 is mapped to FT because the variable "Alram" has the same values in this state. The mv-set (‖ InfoSend ‖ ⊓ ‖ Alarm‖) in (c) is obtained from taking the meet of the values assigned to the states in (a) and (b) according to the truth table to the four-valued lattice in Figure 2.4.

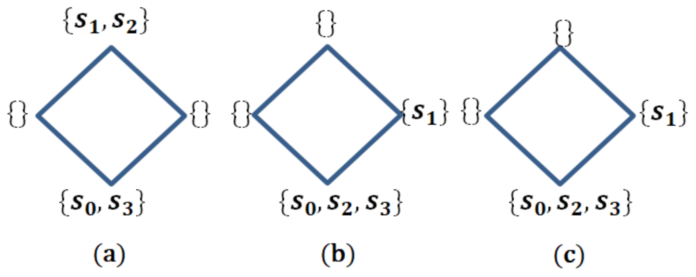


Figure 3.3: (a) 4v-sets of variable InfoSend; (b) 4v-sets of variable Alarm; (c) 4v-sets of (‖InfoSend ‖ ⊓ ‖ Alarm‖)

B. Four-Valued Relations (4v-relations)

A 4v-relation $\mathbb{R}$ on two sets X and Y is a 4v-set on $X \times Y$. Figure 3.4 shows the 4v-relations of model $\mathcal{A}$. The relation from s_0 to s_1 (s_0, s_1) and the other relations are mapped to TT because these relations have the same values in the system. The relation from s_0 to s_3 (s_0, s_3) and the other relations are mapped to FF because these relations have the same values in the system (the relations do not exist). On the other hand, there are no relations with values TF or FT in the system; therefore, these values are mapped with empty sets.

Example 4. To determine the degree of property satisfaction in the 4v-CTL model shown in Figure 3.2 using the multi-valued sets and relations, consider the formula that states *"there exists a path in the system where the formula (InfoSend) holds in the next state". Formally,* $\| \mathit{EX}\ (\mathit{InfoSend}) \|_{s_0}$ where the formula is checked in s_0. By applying the

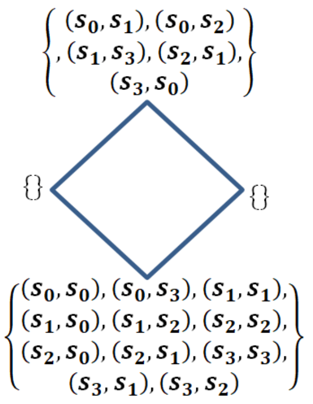

$\{(s_0, s_1), (s_0, s_2), (s_1, s_3), (s_2, s_1), (s_3, s_0)\}$
$\{\}$
$\{\}$
$\{(s_0, s_0), (s_0, s_3), (s_1, s_1), (s_1, s_0), (s_1, s_2), (s_2, s_2), (s_2, s_0), (s_2, s_1), (s_3, s_3), (s_3, s_1), (s_3, s_2)\}$

Figure 3.4: 4v-relation of model $\mathcal{A}$

semantics of $\| EX\varphi \|$ (s) presented in Definition 2.6.3, we conduct the flowing calculation: we take the value of the relation (s_0, s_1) which is TT, meet $(\sqcap)$ with the values of (InfoSend) in s_1 which is TT, then the value of the relation (s_0, s_2) which is TT, meet with the values of (InfoSend) in s_2 which is TT. After that, we take the join $(\sqcup)$ of the obtained results as follows: $(TT \sqcap TT) \sqcup (TT \sqcap TT) = TT \sqcup TT = TT$. It is evident that the result is as expected where the formula (InfoSend) holds with the two designers' agreement in the next states s_1 and s_2. Overall, the result states that the two designers agree that there is at least one next state where the formula holds in the system.

3.4 Reduction Algorithm of 3v-CTL into CTL

As mentioned earlier, we chose the reduction approach to benefit from its advantage of reusing the existing model checkers. We leverage the reduction approach described in [13]. We introduce the formal representation of the algorithm and explain its functionalities.

The reduction algorithm is shown in (Algorithm 1). In this algorithm, the input is the three-valued model $(\mathbb{D}_M)$, and the outputs are the two-valued models D_T, which considers M as True and D_F considers M as False. It gives each two-valued model the same number of states in the original model (line 3) and the same initial state (line 4). In line 5, the state's relations R_T and R_F are initialized by the empty sets so that the algorithm will add these relations as the relations in $\mathbb{D}_M$. In line 6, the algorithm assigns every atomic proposition x in $\mathbb{D}_M$ to the same corresponding state in D_T. Line 7 checks the truth values of x. If it is M, remove x and add two fresh atomic propositions $x-$ and $x+$ with truth values T. Line 8 checks the truth values of x. If it is T, add a fresh atomic proposition $x-$ with truth value F. Line 9 checks the truth values of x. If it is F, add a fresh atomic proposition $x-$ with truth value T.

The same strategy is applied in lines 10-13, but here, every x with value "M" is replaced by two fresh atomic propositions $x-$ and $x+$ with truth values F. Then, in lines 14 to 18, the algorithm gives relations between states (R_T) in the system D_T where these relations are as in system $\mathbb{D}_M$. The same strategy is applied in lines 19 to 23 for assigning the relations between states (R_F) in D_F. Line 24 calls Procedure 1. This procedure converter the input formula ϕ into the positive normal form by pushing all negations to the level of atomic propositions and replacing each negated atomic proposition x in ϕ by $x-$, which is equal to $\neg x$. The procedure then returns the transformed formula ϕ. In lines 25 to 35, the algorithm calls the NuSMV model checker over each two-valued model, and then we compare the results. starting by verifying D_T, if the result is T then check the D_F. If the latter gives T, the final result is T. If it gives F, then the final result is Maybe. If D_T gives F, then the final result is False.

Example 5. In Figure 3.5, the three-valued model $\mathbb{D}_M$ consists of three states with atomic propositions that take truth values over lattice L_3. We need to verify the formulae $(a \wedge \neg b \vee c)$ and $(EX(a \wedge \neg b \vee c))$ starting from s_0 over this model using the reduction algorithm. The main idea is to add an extra variable $x-$ for each atomic proposition x, such that in each state of the model, $x-$ equals $\neg x$. The formula to be verified is converted into the positive normal form by pushing all negations to the level of atomic propositions and replacing each negated atomic proposition x ($\neg x$) with the corresponding proposition ($x-$). Following the algorithm, $\mathbb{D}_M$ is decomposed into two two-valued models. The first is D_T, where atomic proposition b in s_0 is replaced by two atomic propositions b and $b-$, both with truth values T (as b= M and the negation of M is M in $\mathbb{D}_M$). With the atomic proposition a, another atomic proposition is added named $a-$ with value F. For the atomic proposition c, another atomic variable is added named $c-$ with value T. The same strategies are applied to the atomic propositions in states s_1 and s_2. On the other hand, in D_F, the same atomic proposition b in s_0 is replaced by two atomic propositions b and $b-$ both with truth values F (here we consider M as false). The atomic propositions a and c are treated as in D_T. The same strategies are applied to the atomic propositions in states s_1 and s_2 from D_F. By doing so, we obtain two-valued models with truth values T and F, which make them suitable to be verified by NuSMV checker. In $\mathbb{D}_M$, it is unknown whether the two formulae are satisfied because we don't know about b and c in s_0 and s_1. We can derive and prove the verification results after transforming $\mathbb{D}_M$ to CTL models and replacing every negated proposition $\neg x$ in the formula by $x-$, it is evident that the first formula is verified by value T and the other by F. Therefore, the final result is Maybe. The second formula is verified by T in both models. Therefore, the final result is True.

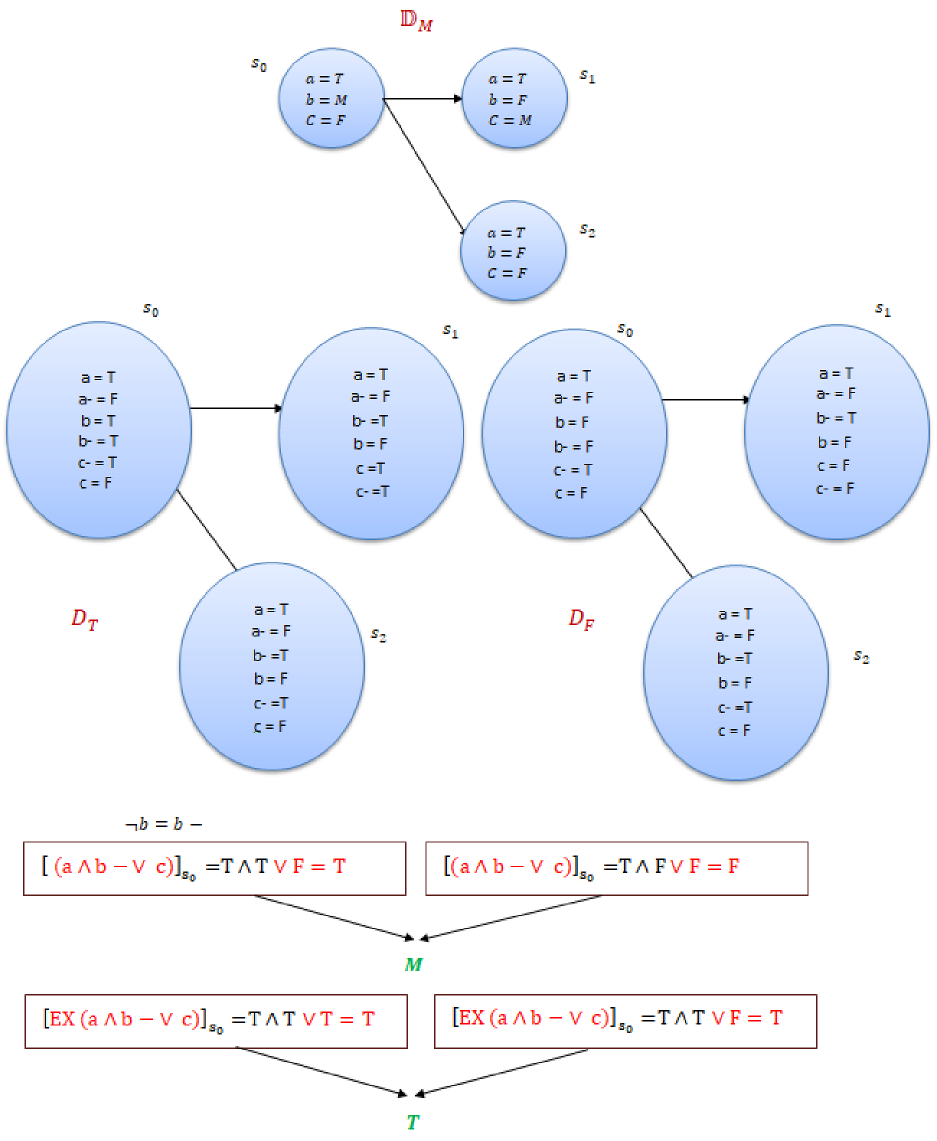


Figure 3.5: Transforming the 3v-model $\mathbb{D}_M$ into two two-valued models D_T and D_F

Algorithm 1 Transform $\mathbb{D}_M = (S_M, R_M, I_M, AP_M, \mathbb{O}_M)$ into two-valued $D_T = (S_T, R_T, I_T, AP_T, L_T)$ and two-valued $D_F = (S_F, R_F, I_F, AP_F, L_F)$

```
1: **Input** the model $\mathbb{D}_M$ and $\phi$.
2: **Output** the models $D_T$, $D_F$ and a formula $\phi_1$
3: $S_T = S_M$ and $S_F = S_M$
4: $I_T = I_M$ and $I_F = I_M$
5: Initialize $R_T = \emptyset$ and $R_F = \emptyset$
6: Initialize $(L_T(S_T))(x) = (\mathbb{O}_M(S_M))(x)$
7: For every $(L_T(S_T))(x) = M \Rightarrow$ replace $(L_T(S_T))(x)$ by $(L_T(S_T))(x) = T$ and add $(L_T(S_T))(x-) = T$
8: For every $(L_T(S_T))(x) = T \Rightarrow$ add $(L_T(S_T))(x-) = F$
9: For every $(L_T(S_T))(x) = F \Rightarrow$ add $(L_T(S_T))(x-) = T$
10: Initialize $(L_F(S_F))(x) = (\mathbb{O}_M(S_M))(x)$
11: For every $(L_F(S_F))(x) = M \Rightarrow$ replace $(L_F(S_F))(x)$ by $(L_F(S_F))(x) = F$ and add $(L_F(S_F))(x-) = F$
12: For every $(L_F(S_F))(x) = T \Rightarrow$ add $(L_F(S_F))(x-) = F$
13: For every $(L_F(S_F))(x) = F \Rightarrow$ add $(L_F(S_F))(x-) = T$
14: **for each** $(s_M, s_M^{'}) \in S_M^2$ **do**
15:     **if** $(s_M, s_M^{'}) \in R_M$ **then**
16:         $R_T := R_T \cup \{(s_T, s_T^{'})\}$
17:     **end if**
18: **end for**
19: **for each** $(s_M, s_M^{'}) \in S_M^2$ **do**
20:     **if** $(s_M, s_M^{'}) \in R_M$ **then**
21:         $R_F := R_F \cup \{(s_F, s_F^{'})\}$
22:     **end if**
23: **end for**
24: $(\phi_1) \leftarrow$ TRANSFORMULA$(\phi)$
25: **Call NuSMV on** $D_T$
26: **if** $D_T \not\models \phi_1$ **then**
27:     "Formula is False"
28: **else**
29:     **Call NuSMV on** $D_F$
30:     **if** $D_F \models \phi_1$ **then**
31:         "Formula is True"
32:     **else**
33:         "Formula is M"
34:     **end if**
35: **end if**
```

Procedure 1 TRANSFORM FORMULA ϕ

```
procedure TRANSFORMULA($\phi$)
    Transform $\phi$ to the positive normal form
    for each $x$ do
        if $\neg x$ then
            $\neg x = x-$
        end if
    end for
    $\phi_1 = \phi$
    Return $\phi_1$
end procedure
```

3.4.1 Soundness of Algorithm 1

Theorem 1. *(Soundness of Algorithm 1): Let $\mathbb{D}_M$ and ϕ be a 3v-CTL model and a formula to be verified over this model. Also, let D_T and D_F be the corresponding CTL models, and ϕ_1 be the transformed formula to be verified over both models.*
We have:

1. *$\parallel \phi \parallel_{\mathbb{D}_M} = T$ iff $D_T \models \phi_1$ and $D_F \models \phi_1$, means the satisfaction degree of ϕ in the three-valued model $\mathbb{D}_M$ is true iff the corresponding formula is satisfied in both models.*

2. *$\parallel \phi \parallel_{\mathbb{D}_M} = F$ iff $D_T \not\models \phi_1$ and $D_F \not\models \phi_1$, means the satisfaction degree of ϕ in $\mathbb{D}_M$ is false iff the corresponding formula is unsatisfied in both models.*

3. *$\parallel \phi \parallel_{\mathbb{D}_M} = M$ iff $D_T \models \phi_1$ and $D_F \not\models \phi_1$, means the satisfaction degree of ϕ in $\mathbb{D}_M$ is uncertain iff the corresponding formula is satisfied in the first model and not satisfied in the second one.*

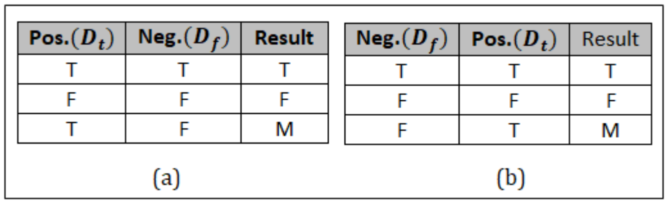
Proof. The proof of this theorem is straightforward, using induction with respect to the formula. Let's prove these statements one by one:

1. According to the truth table (Figure 3.6, (a)) built from our algorithm (lines 25-35), if ϕ_1 is satisfied in D_T and D_F then ϕ (the original formula) will be assigned the truth value "true" in $\mathbb{D}_M$. This proves the first point.

2. Let the same formula ϕ_1 be verified on D_T and D_F. According to the same truth table, if ϕ_1 is not satisfied in D_T and D_F then ϕ (the original formula) will not be satisfied by taking the truth value "false" in $\mathbb{D}_M$. This proves the second point.

3. On the other hand, if ϕ_1 is satisfied in D_T and not satisfied in D_F, then the original formula ϕ will be assigned the truth value "Maybe". This proves the third point.

The same applies to the three points if the algorithm starts by verifying D_F where we have the truth table shown in Figure 3.6(b). □



Pos.(D_t)	Neg.(D_f)	Result
T	T	T
F	F	F
T	F	M

(a)

Neg.(D_f)	Pos.(D_t)	Result
T	T	T
F	F	F
F	T	M

(b)

Figure 3.6: (a) The truth table of algorithm 1 when it starts by verifying D_T and (b) the truth table when it starts by verifying D_F

Chapter 4

Modeling and Verifying IoT/IS Systems with mv-Commitment Logics

4.1 Overview and Motivation

Figure 4.1 gives the overview of the work performed on multi-valued commitment logics. The main goal is to provide practical and reliable approaches for model-checking IoT and IS with flexible commitment protocols under uncertain or inconsistent settings. We start by showing that the main base of our logic is the Computational Tree logic (CTL) presented in [43]. CTL was extended in [31] to the logic of commitment CTL^{cc} and in [17] to the mv-CTL mv-CTL. Based on these logics, we extended the mv-CTL logic to 3v-CTL^{cc}, the three-valued logic of conditional commitment in [8], and unconditional commitment 3v-CTL^{c} in [3]. We are motivated to present effective approaches to reason about uncertainty over IoT/IS with commitment protocols. Moreover, we expanded these logics to new versions named 4v-CTL^{cc} and 4v-CTL^{c} to represent affective approaches to reason about inconsistency over the systems under consideration.

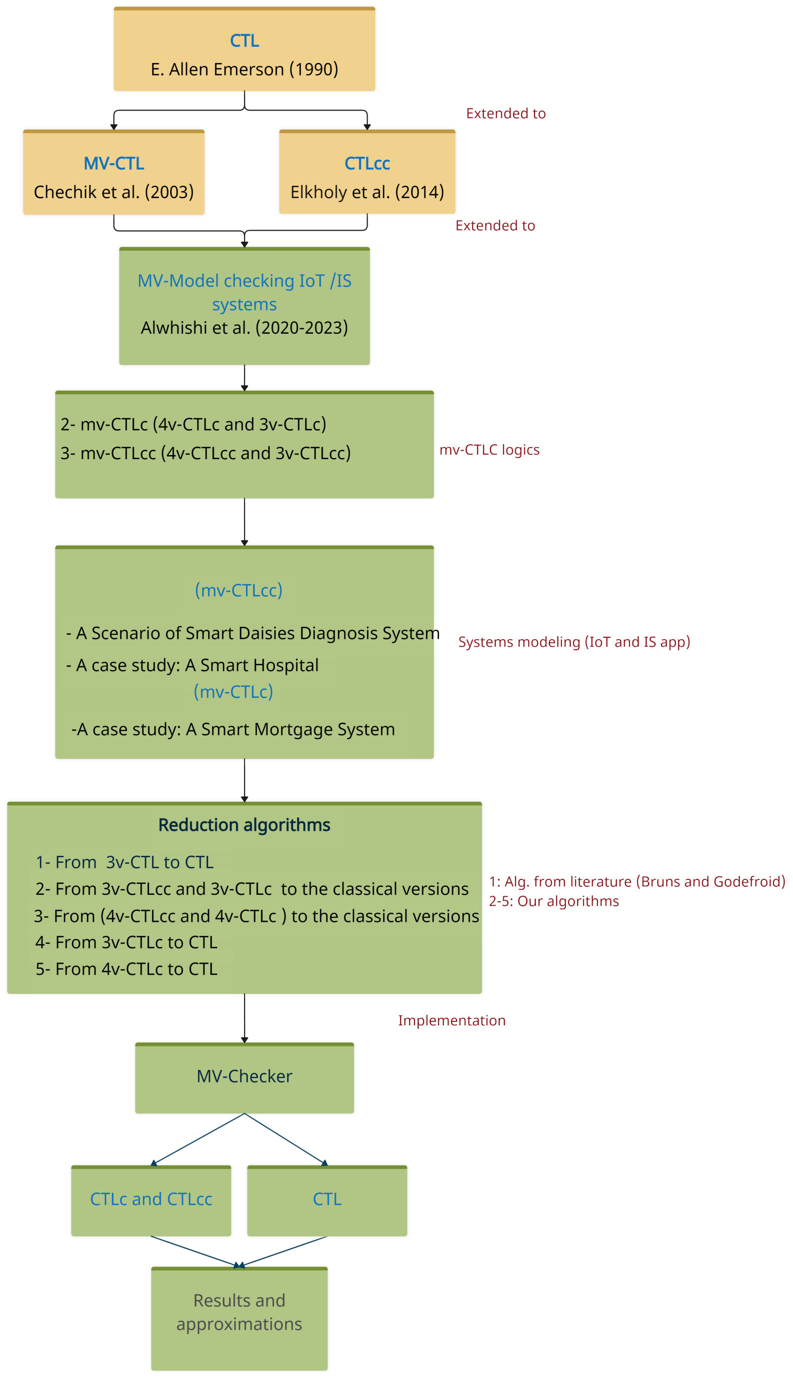

CTL
E. Allen Emerson (1990)
Extended to
MV-CTL
Chechik et al. (2003)
CTLcc
Elkholy et al. (2014)
Extended to
MV-Model checking IoT /IS systems
Alwhishi et al. (2020-2023)
2- mv-CTLc (4v-CTLc and 3v-CTLc)
3- mv-CTLcc (4v-CTLcc and 3v-CTLcc)
mv-CTLC logics
(mv-CTLcc)
- A Scenario of Smart Daisies Diagnosis System
- A case study: A Smart Hospital
(mv-CTLc)
-A case study: A Smart Mortgage System
Systems modeling (IoT and IS app)
Reduction algorithms
1- From 3v-CTL to CTL
2- From 3v-CTLcc and 3v-CTLc to the classical versions
3- From (4v-CTLcc and 4v-CTLc) to the classical versions
4- From 3v-CTLc to CTL
5- From 4v-CTLc to CTL
1: Alg. from literature (Bruns and Godefroid)
2-5: Our algorithms
Implementation
MV-Checker
CTLc and CTLcc
CTL
Results and approximations

Figure 4.1: The chapter overview

The multi-valued logics presented in this work effectively apply to any commitment-based multi-agent system with uncertainty or inconsistency. However, we focus on applying these logics over IS and IoT systems because of their high susceptibility to uncertainty and inconsistency. We model a specific scenario of *a Smart Daisies Diagnosis application* and *a Smart Hospital application* using our 3v-CTL^{cc} and 4v-CTL^{cc}. Furthermore, we model *A Smart Mortgage application* using both 3v-CTL^{c} and 4v-CTL^{c} for reasoning about uncertainty and inconsistency.

To the best of our knowledge, there are no tools that directly deal with the multi-valued CTL model checking problems. Motivated by the advantages of reusing the existing model checking tools, we choose the reduction techniques for transforming the multi-valued commitment logics to their two-valued versions and CTL. We introduce new reduction algorithms. **Algorithm 2** transforms 3v-CTL^{c} and 3v-CTL^{cc} to their two-valued versions CTL^{c} and CTL^{cc} respectively. Similarly, **Algorithm 3** transforms 4v-CTL^{c} and 4v-CTL^{cc} to their respective two-valued versions. **Algorithm 4** transforms 3v-CTL^{c} to CTL and **Algorithm 5** transforms the 4v-CTL^{c} to CTL. For each algorithm, we provide soundness proofs and intensive computational complexity analysis.

We develop two new Java-based tools named **MV-Checker** for implementing the algorithms. The tool transforms the multi-valued logics to CTL and automatically interacts with the NuSMV model checker. It also transforms these logics into their two-valued cases and automatically interacts with the MCMAS+ model checker used for multi-agent systems with commitment protocols. The tool is built on top of the tool developed by [25]. Finally, we report and compare our findings with detailed discussions in four different cases studies to show that our logics and model checking algorithms apply to various multi-agent commitment systems.

4.2 Modeling Uncertainty in IoT Systems with Three-Valued Conditional Commitments (3v-CTL^{cc})

4.2.1 3v-CTL^{cc}

In this section, we recall our work in [8], where we introduced a new logic named 3v-CTL^{cc} for modeling and verifying IoT applications with timed conditional commitment protocols under uncertainty. This logic is an extension to the three-valued case of mv-CTL explained in Section 2.6 where it is based on the three-valued lattice with truth values (T, M, F) explained in Section 2.5.1. Our logic is produced for capturing the conditional commitment, explained in Section 2.2, among the system's agents, but here we deal with missing information about some system's behaviors. In this logic, the system is modeled based on the 3v-CTL^{cc} where the atomic promotions (formulae) take truth values T, M or F. The value M presents the messing or uncertain information. For example, consider the system model shown in Figure 4.2. This model represents the same scenario of a smart home explained in Section 2.2. Assume we have the conditional commitment formula *"When the user sets a specific washing time, the smart washing machine commits to washing the clothes within this specified time"*. The first part of this formula represents the condition ψ in the accessible state s_1 with the assumption that the system designer misses information about this condition, whether it holds in s_1 or not. The second part represents ϕ, which is true in the same state. Based on this, the satisfaction degree of the formula $\parallel CC_{i\to j}(\psi, \varphi) \parallel (s_0)$ re-expressed by $\parallel CC_{washingmachine\to user}(\text{UserSetsTime,Washing}) \parallel (s_0)$ is computed as $M \sqcap T = M$. According to the semantics, ψ and ϕ should hold together in the same accessible state.

4.2.2 3v-CTL^{cc} IoT model

The model is obtained from the two-valued model of CTL^{cc} by extending the latter with the lattice L_3 and replacing the valuation function V by the multi-valuation function $\mathbb{O} : S \to (\text{AP} \to L_3)$, a total labeling function which maps every atomic proposition $a \in AP$ in

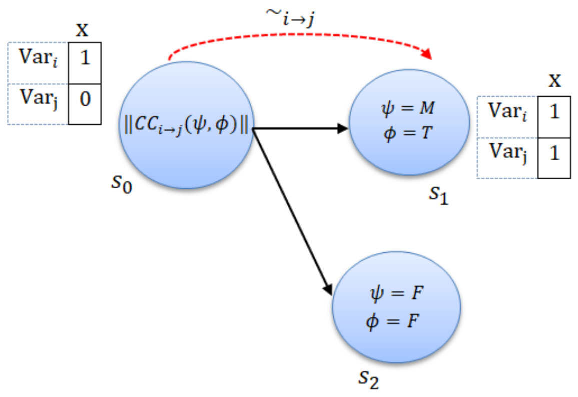


Figure 4.2: An example of conditional commitment model with uncertainty

$s \in S$ to L_3. Thus, $(\mathbb{O}(s))(a) = l$ means the atomic variable a has value l from L_3 in state s where $a \in$ AP.

4.2.3 Syntax

The syntax of 3v-CTL^{cc} is equivalent to CTL^{cc}, except that formulae are evaluated over the three-valued lattice.

4.2.4 Semantics

The semantics of this logic is an extension to (mv-CTL) considering that we deal with L_3. Below, we add our semantics of the 3v-CTL^{cc}.

Given a model of 3v-CTL^{cc} ($\mathbb{K}_M$) and a conditional commitment formula, the satisfaction degree of this formula is defined as:

- $\| CC_{i\to j}(\psi, \varphi) \| \; (s) = T$ *iff* (1) $\exists s' \in S$ *s.t.* $s \sim_{i\to j} s'$ *and* $\| \psi \| \; (s') = T$ *and* (2) $\forall s' \in S s.t.$ $s \sim_{i\to j} s'$ *and* $\| \psi \| \; (s') = T$ we have $\| \phi \| \; (s') = T$. Which means, the satisfaction degree of the formula $CC_{i\to j}(\psi, \varphi)$ in state s is “true” if the truth degree of ψ and ϕ in the accessible state s' is T;

- $\| CC_{i\to j}(\psi, \varphi) \| \; (s) = M$ *iff*

  - (1) $\exists s' \in S s.t.$ $s \sim_{i\to j} s'$ *and* $\| \psi \| \; (s') = M$ *and*
    (2) $\forall s' \in S s.t.$ $s \sim_{i\to j} s'$ *and* $\| \psi \| \; (s') = M$ we have $\| \phi \| \; (s') = M \vee T$

This means the satisfaction degree of the given conditional commitment formula $CC_{i\to j}(\psi,\varphi)$ in state s is M if the truth degree of ψ in the $s^{'}$ is M and the truth degree of ϕ is M or T as $M \sqcap M = M$ and $M \sqcap T = M$;

OR

- (1) $\exists s^{'} \in S s.t.\ s \sim_{i\to j} s^{'} and \parallel \psi \parallel (s^{'}) = T\ \ and$

  (2) $\forall s^{'} \in S s.t.\ s \sim_{i\to j} s^{'} and \parallel \psi \parallel (s^{'}) = T$ we have$\parallel \phi \parallel (s^{'}) = M$. This means the satisfaction degree of the conditional commitment formula $CC_{i\to j}(\psi,\varphi)$ in state s is M if the truth degree of ψ in the $s^{'}$ is T and the truth degree of ϕ is M as $T \sqcap M = M$.

Example 6. Modeling a scenario of a Smart Daisies Diagnosis system with 3v-CTL^{cc}: Consider the simple scenario of *a Smart Daisies Diagnoses system* shown in Figure 4.3. The system starts at state s_0 where the symptoms (*SympCol*) are collected using an intelligent algorithm that sends the diagnosis (*Diag*) to state s_1 for performing a smart diagnosis process. Then, the application displays at states s_2 and s_3 multiple diseases (*Dise1* and *Dise2*) according to the generated diagnosis. The atomic propositions over this system take truth values over lattice L_3 for interpreting the uncertainty. This system contains a conditional commitment interaction between the application and the physician. This interaction is captured by the commitment accessibility relation from s_1 to s_2 indicated by color red. Focusing on conditional commitment, the proposition *Dise1* in s_2 has uncertain information because we miss information about the *Dise1*. This missing information could be occurred because of system abstraction or partition. Therefore, we have uncertainty regarding the satisfaction of the conditional commitment formula "*The smart system commits to displaying disease 1 for the physician when the diagnosis is received*", logically expressed by $CC_{System\to Physician}(Diag, dise1)$.

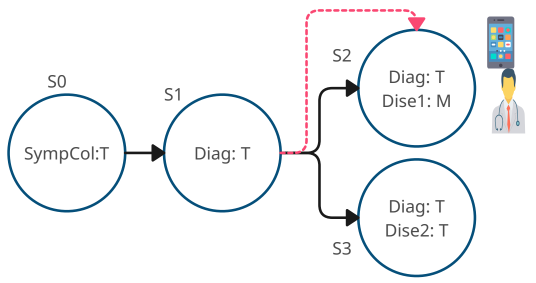


Figure 4.3: A scenario of a Smart Daisies Diagnoses system

4.3 Modeling Inconsistency in IoT/IS Systems with Four-Valued Conditional Commitments (4v-CTL^{cc}

4.3.1 4v-CTL^{cc}

We introduce this logic to facilitate the modeling and verification of IoT and IS applications with commitment, especially when the same system is designed by multiple designers with different viewpoints regarding system behaviors. Specifically, this logic allows the verification results to indicate who asserts what regarding specific commitment formulae. The logic is based on the four-valued lattice (L_4) explained in Section 2.5.2, where the system under investigation is modeled, and the atomic propositions in corresponding states take truth values TT, FF, FT or TF, following the same strategy applied in Example 2, Section 3.

4.3.2 4v-CTL^{cc} IoT model

The model is obtained from the two-valued model of CTL^{cc} by extending the latter with the lattice L_4 and replacing the valuation function V by the multi-valuation function $\mathbb{O}$: $S \rightarrow (\text{AP} \rightarrow L_4)$, a total labeling function which maps every atomic proposition $a \in AP$ in $s \in S$ to L_4.

4.3.3 Syntax

The syntax of 4v-CTL^{cc} is equivalent to CTL^{cc}, except that formulae are evaluated over the four-valued lattice.

4.3.4 Semantics

The semantics of this logic is an extension to (mv-CTL) considering that we deal with L_4. Below, we add our semantics of the 4v-CTL^{cc}.

Given a model of 4v-CTL^{cc} ($\mathbb{H}_c$) and a conditional commitment formula, the satisfaction degree of this formula is defined as:

- $\| CC_{i \to j}(\psi, \varphi) \| (s) = TT$ *iff* (1) $\exists s^{'} \in S$ *s.t.* $s \sim_{i \to j} s^{'} and \parallel \psi \parallel (s^{'}) = TT$ *and* (2) $\forall s^{'} \in S s.t.$ $s \sim_{i \to j} s^{'} and \parallel \psi \parallel (s^{'}) = TT$ we have $\parallel \phi \parallel (s^{'}) = TT$. This means the satisfaction degree of the formula $CC_{i \to j}(\psi, \varphi)$ in state s is "true, true" (positive agreement) if the truth degree of ψ and ϕ in the accessible state $s^{'}$ is TT;

- $\parallel CC_{i \to j}(\psi, \varphi) \parallel (s) =$ TF iff

  - (1) $\exists s^{'} \in S s.t.$ $s \sim_{i \to j} s^{'} and \parallel \psi \parallel (s^{'}) =$TF *and*
    (2) $\forall s^{'} \in S s.t.$ $s \sim_{i \to j} s^{'} and \parallel \psi \parallel (s^{'}) =$ TF we have$\parallel \phi \parallel (s^{'}) =$TF $\vee$ TT
    This means the satisfaction degree of the given conditional commitment formula $CC_{i \to j}(\psi, \varphi)$ in state s is TF if and only if the truth degree of ψ in the $s^{'}$ is TF and the truth degree of ϕ is TF or TT as TF $\sqcap$ TT =TF and TF $\sqcap$ TF=TF ;
    OR

  - (1) $\exists s^{'} \in S s.t.$ $s \sim_{i \to j} s^{'} and \parallel \psi \parallel (s^{'}) =$ TT *and*
    (2) $\forall s^{'} \in S s.t.$ $s \sim_{i \to j} s^{'} and \parallel \psi \parallel (s^{'}) =$ TT we have$\parallel \phi \parallel (s^{'}) =$TF. This means the satisfaction degree of the conditional commitment formula $CC_{i \to j}(\psi, \varphi)$ in state s is TF if the truth degree of ψ in the $s^{'}$ is TT and the truth degree of ϕ is TF as TT $\sqcap$ TF =TF;

- $\parallel CC_{i \to j}(\psi, \varphi) \parallel (s) =$ FT iff

- (1) $\exists s^{'} \in Ss.t.\ s \sim_{i \to j} s^{'} and \parallel \psi \parallel (s^{'})$ =FT *and*

  (2) $\forall s^{'} \in Ss.t.\ s \sim_{i \to j} s^{'} and \parallel \psi \parallel (s^{'})$ = FT we have$\parallel \phi \parallel (s^{'})$ =TF $\vee$ TT

  This means the satisfaction degree of the given conditional commitment formula $CC_{i \to j}(\psi, \varphi)$ in state s is FT if and only if the truth degree of ψ in the $s^{'}$ is FT and the truth degree of ϕ is FT or TT as FT $\sqcap$ TT =FT and FT$\sqcap$ FT=FT

  OR

- (1) $\exists s^{'} \in Ss.t.\ s \sim_{i \to j} s^{'} and \parallel \psi \parallel (s^{'})$ = TT *and*

  (2) $\forall s^{'} \in Ss.t.\ s \sim_{i \to j} s^{'} and \parallel \psi \parallel (s^{'})$ = TT we have$\parallel \phi \parallel (s^{'})$ =FT. This means the satisfaction degree of the conditional commitment formula $CC_{i \to j}(\psi, \varphi)$ in state s is FT if the truth degree of ψ in the $s^{'}$ is TT and the truth degree of ϕ is FT as TT $\sqcap$ FT =FT.

Example 7. Modeling a scenario of Smart Diagnosis system with 4v-CTL^{cc}:
Considering the same system in Figure 4.3. We assume two experts design the system; therefore, the state variables take truth values between TT, FF, TF and FT. For example, in state s_2 *Dise1: TF* means, the first expert says "yes" and the second says "no" about holding this atomic proposition in this state. In this case, we have inconsistency in verifying the same commitment formula $CC_{System \to Physician}(Diag, Dise1)$.

4.4 Modeling Uncertainty in IoT/IS with Three-Valued Unconditional Commitments (3v-CTL^c)

4.4.1 3v-CTL^c

This section presents the 3v-CTL^c, our logic for verifying IoT/IS applications with timed unconditional commitment protocols under uncertainty settings introduced in [3]. It differs from the three-valued conditional commitment explained in Section 4.2, where the focus is on systems with missing information when the commitment between agents does not emphasize the condition ψ. In other words, agent i commits to bringing about a particular formula ϕ for agent j, without any specific condition that must be satisfied by the latter. For example, in a smart health monitoring system, we need to verify an unconditional commitment formula stating that *"the temperature sensor commits to sending information to the patient via an application installed on their smartphone."* We assume that it is uncertain whether the formula *"sending information"* holds in a particular state in the system. The verification of such a formula cannot be performed with the two-valued unconditional commitment logic. Therefore the 3v-CTL^c is developed.

4.4.2 3v-CTL^c IS model

The model extends from the CTL^c by considering the lattice structure $(L_3, \sqcup, \sqcap)$. In this model, the valuation V is replaced by $\mathbb{O} : S \to (AP \to L_3)$ defined as in the previous section.

4.4.3 Syntax

The syntax of this logic is defined as the syntax of CTL^c, except that formulae are evaluated over the three-valued lattice.

4.4.4 Semantics

Following the same strategy with 3v-CTL^{cc}, we add our three-valued modalities for the unconditional commitment logic:

- $\parallel C_{i\to j}(\varphi) \parallel_{\mathbb{U}} (s) = T$ *iff* $\forall s^{'} \in S$ *s.t.* $s \sim_{i\to j} s^{'}$ we have $\parallel \varphi \parallel_{\mathbb{U}} (s^{'}) = T$;

  This means the satisfaction degree of the unconditional formula $C_{i\to j}(\varphi)$ in state s of the system $\mathbb{U}$ is "true" if and only if the truth degree of formula φ in all the accessible states $s^{'}$ is T;

- $\parallel C_{i\to j}(\varphi) \parallel_{\mathbb{U}} (s) = M$ *iff*

  $\forall s^{'} \in S$ *s.t.* $s \sim_{i\to j} s^{'}$ we have $\parallel \varphi \parallel_{\mathbb{U}} (s^{'}) \neq F$ and $\exists s^{'} \in S$ *s.t.* $s \sim_{i\to j} s^{'}$ and $\parallel \varphi \parallel_{\mathbb{U}} (s^{'}) = M$.

  This means the satisfaction degree of the unconditional commitment formula is M if and only if the truth degree of formula φ in every $s^{'}$ is not equal to F and there is at least one $s^{'}$ which holds φ with M. In other words, the φ in all the accessible states must be T or M because $T \sqcap M \; = M$ and must not be F where $F \sqcap M \sqcap T = F$.

4.5 Modeling Inconsistency in IS/IoT with Four-Valued Unconditional Commitments (4v-CTL^c)

4.5.1 4v-CTL^c)

This section introduces a new modality for unconditional commitment logic under inconsistency settings. We follow the same strategy applied to the four-valued conditional commitment in Section 4.3, except here we do not empathize on the condition ψ. In particular, this logic deals with systems with inconsistency where the commitment among the system's agents i and j is expressed as *agent i commits to bringing about ϕ* for agent j without a condition must be satisfied by the latter.

4.5.2 4v-CTL^c IS model

The model extends from the CTL^c by considering the lattice structure $(L_4, \sqcup, \sqcap)$. In this model, the valuation V is replaced by $\mathbb{O} : S \rightarrow (AP \rightarrow L_4)$ defined as in the previous section.

4.5.3 Syntax

The syntax of this logic is defined as the syntax of CTL^c, except that formulae are evaluated over the four-valued lattice.

4.5.4 Semantics

The semantics of this logic is an extension to the four-valued case of mv-CTL. Below, we added our four-valued modalities for the unconditional commitment logic: Let $\mathbb{Y}_c$ be a 4v-CTL^c model, then:

- $\parallel C_{i\to j}(\varphi) \parallel_{\mathbb{Y}_c} (s) = TT$ *iff* $\forall s^{'} \in S$ *s.t.* $s \sim_{i\to j} s^{'}$ we have $\parallel \varphi \parallel_{\mathbb{Y}_c} (s^{'}) = TT$;

  This means the satisfaction degree of the unconditional commitment formula $C_{i\to j}(\varphi)$ in state s of the 4v-system $\mathbb{Y}_c$ is TT if and only if the truth degree of the formula φ in all $s^{'}$ is TT. i.e there is agreement about the satisfaction of the formula;

- $\parallel C_{i\to j}(\varphi) \parallel_{\mathbb{Y}_c} (s) = TF$ *iff* $\forall s^{'} \in S$ *s.t.* $s \sim_{i\to j} s^{'}$ we have $\parallel \varphi \parallel_{\mathbb{Y}_c} (s^{'}) \neq FF$ and $\exists s^{'} \in S$ *s.t.* $s \sim_{i\to j} s^{'}$ and $\parallel \varphi \parallel_{\mathbb{Y}_c} (s^{'}) = TF$;

  This means the satisfaction degree of the commitment formula is TF if and only if the truth degree of φ in all $s^{'}$ is not equal to FF and there is at least one $s^{'}$ that holds φ with the value TF. in other words, the values of φ in all $s^{'}$ must be at least TF as $TT \sqcap TF = TF$ and must not include FF as $FF \sqcap TF \sqcap TT = FF$;

- $\parallel C_{i\to j}(\varphi) \parallel_{\mathbb{Y}_c} (s) = FT$ *iff* $\forall s^{'} \in S$ *s.t.* $s \sim_{i\to j} s^{'}$ we have $\parallel \varphi \parallel_{\mathbb{Y}_c} (s^{'}) \neq FF$ and $\exists s^{'} \in S$ *s.t.* $s \sim_{i\to j} s^{'}$ and $\parallel \varphi \parallel_{\mathbb{Y}_c} (s^{'}) = FT$.

  This semantics means the satisfaction degree of the formula $C_{i\to j}(\varphi)$ in state s of the system $\mathbb{Y}_c$ is FT if and only if the truth degree of φ in all the accessible states $s^{'}$ is not equal to FF and there at least one accessible state $s^{'}$ that holds φ with the value FT. Intuitively, the values of φ in the accessible states must include at least FT as $TT \sqcap FT = FT$ and must not include FF as $FF \sqcap FT \sqcap TT = FF$.

4.6 Reduction-Based Multi-Valued Model Checking mv-CTL^{cc} and mv-CTL^{c}

4.6.1 Reduction Algorithm from 3v-CTL^{cc} to CTL^{cc}

We introduce a new algorithm (Algorithm 2) for transforming our logic 3v-CTL^{cc} to its two-valued version CTL^{cc} to reuse the existing model checker MCMAS+. Algorithm 2 starts by taking the 3v-CTL^{cc} model ($\mathbb{K}_M$) and the commitment formula as inputs in line 1. Line 2, call Procedure 1 to convert the negation of the formula into the level of an atomic proposition as explained in Example 5. Line 3 shows that the algorithm outputs two two-valued CTL^{cc} models K_T and K_F. Line 4 gives the two two-valued models the same number of states and initial states as the 3v-model. Line 5 initializes the transitions and commitment accessibility relations by empty sets to start assigning the same transitions and relations to the two-valued models. R_T and AR_T represent the transitions and commitment accessibility relations (explained in Example1) in K_T, respectively. R_F and AR_F represent the transitions and commitment accessibility relations in K_F, respectively. Line 6 gives the same atomic positions in $\mathbb{K}_M$ to the two models. Line 7 replaces every proposition x with value M by $x = T$ and adds a fresh atomic proposition $x- = T$. Line 8 adds for every atomic proposition $x = T$ another atomic position $x- = F$. Line 9 adds for every atomic proposition $x = F$ another atomic position $x- = T$. The same strategy is applied in lines 10-13, but here, it replaces every proposition x with value M by $x = F$ and adds a fresh atomic proposition $x- = F$. Line 14 gets the set of states for each model with the transformed atomic propositions. Through lines (15-22), the algorithm starts, through the procedure TRANSFORM, to relate the states of the two-valued model K_T similar to the transition and accessibility relations in the 3v-model. The same applies to K_F through lines (23-29). Line 30 returns the two complete two-valued models, and line 31 ends the procedure. The algorithm calls MCMAS+ over the two-valued CTL^{cc} models and the input formula in lines (32-33) and saves the results in z and y in line 34. Then, through lines (35-42), applies the approximation based on the join-irreducible elements in L_3 by taking the join of the verification results where the result (False) is considered $\emptyset$. We change the value F to an empty set because the final result is computed by taking into consideration only the two-valued model with the verification result True. Otherwise, we exclude the model. Specifically, $M \sqcup \emptyset =$ Maybe means we take the verification result of the model that considers M as True (M and T are true) and exclude the result of the model that considers M as False (only T is true).

Algorithm 2 Transform $\mathbb{K}_M = (S_M, R_M, \{\sim_{i\to j} | \ (i,j) \in A^2\}, I_M, \mathbb{O}_M)$ into two-valued $K_T = (S_T, R_T, \{\sim_{i\to j} | \ (i,j) \in A^2\}, I_T, L_T)$ and two-valued $K_F = (S_F, R_F, \{\sim_{i\to j} | \ (i,j) \in A^2\}, I_F, L_F)$

```
1: **Input** the mv-model $\mathbb{K}_M$ and a commitment formula $\phi = CC_{i\to j}(\psi, \varphi)$
2: $\phi_1 \leftarrow$ TRANSFORMULA($\phi$)
3: **Output** the two-valued models $K_T$ and $K_F$
4: $S_T = S_M$, $S_F = S_M$, $I_T = I_M$ and $I_F = I_M$
5: Initialize $R_T = \emptyset$, $R_F = \emptyset$, $AR_T = \emptyset$ and AR$_F = \emptyset$
6: Initialize $(L_T(S_T))(x) = (\mathbb{O}_M(S_M))(x)$
7: For every $(L_T(S_T))(x) = M \Rightarrow$ replace $(L_T(S_T))(x)$ by $(L_T(S_T))(x) = T$ and add $(L_T(S_T))(x-) = T$
8: For every $(L_T(S_T))(x) = T \Rightarrow$ add $(L_T(S_T))(x-) = F$
9: For every $(L_T(S_T))(x) = F \Rightarrow$ add $(L_T(S_T))(x-) = T$
10: Initialize $(L_F(S_F))(x) = (\mathbb{O}_M(S_M))(x)$
11: For every $(L_F(S_F))(x) = M \Rightarrow$ replace $(L_F(S_F))(x)$ by $(L_F(S_F))(x) = F$ and add $(L_F(S_F))(x-) = F$
12: For every $(L_T(S_T))(x) = T \Rightarrow$ add $(L_T(S_T))(x-) = F$
13: For every $(L_T(S_T))(x) = F \Rightarrow$ add $(L_T(S_T))(x-) = T$
14: **Get** the sets $S_T$ and $S_F$
15: **procedure** TRANSFORM $(S_T, S_F, \mathbb{K}_M)$
16: **for each** $(s_M, s_M^{'}) \in S_M^2$ **do**
17:     **if** $(s_M, s_M^{'}) \in R_M$ **then**
18:         $R_T := R_T \cup \{(s_M, s_M^{'})\}$
19:         **if** $s_M \sim_{i\to j} s_M^{'}$ **then** AR$_T$ =AR$_T \cup \{s_M \sim_{i\to j} s_M^{'}\}$
20:         **end if**
21:     **end if**
22: **end for**
23: **for each** $(s_M, s_M^{'}) \in S_M^2$ **do**
24:     **if** $(s_M, s_M^{'}) \in R_M$ **then**
25:         $R_F := R_F \cup \{(s_M, s_M^{'})\}$
26:         **if** $s_M \sim_{i\to j} s_M^{'}$ **then** AR$_F$ =AR$_F \cup \{s_M \sim_{i\to j} s_M^{'}\}$
27:         **end if**
28:     **end if**
29: **end for**
30: **Return** $(K_T, K_F)$
31: **EndProcedure**
32: **Call MCMAS+ on** $K_T$ **and** $\phi_1$
33: **Call MCMAS+ on** $K_F$ **and** $\phi_1$
34: $z :=$ MCMAS result on $K_T$ and $\phi_1$ and $y :=$ MCMAS result on $K_F$ and $\phi_1$
35: **if** $z =$ True and $y =$ True **then** the final result: $M \sqcup T =$ True
36: **else**
37:     **if** $z =$ True and $y =$ False **then** the final result: $M \sqcup \emptyset$ =Maybe
38:     **else**
39:         **if** $z =$ False and $y =$ False **then** the final result: $\emptyset \sqcup \emptyset = \emptyset = False$
40:         **end if**
41:     **end if**
42: **end if**
```

4.6.2 Soundness of Algorithm 2

Soundness of Algorithm 2

Theorem 2. *(Soundness of Algorithm 2) Let $\mathbb{K}_M$ be a 3v-CTL^{cc} model and $CC_{i\to j}(\psi,\varphi)$ a conditional commitment formula to be verified over this model. Also, let K_T and K_F be the corresponding two-valued CTL^{cc} models, and ϕ_1 be the transformed formula to be verified over both models.*
We have:

1. *$\| CC_{i\to j}(\psi,\varphi) \|_{\mathbb{K}_M} = T$ iff $K_T \models \phi_1$ and $K_F \models \phi_1$, means the satisfaction degree of the formula in the three-valued model $\mathbb{K}_M$ is true iff the transformed formula is satisfied in both models.*

2. *$\| CC_{i\to j}(\psi,\varphi) \|_{\mathbb{K}_M} = F$ iff $K_T \not\models \phi_1$ and $K_F \not\models \phi_1$, means the satisfaction degree of the formula in $\mathbb{K}_M$ is false iff the transformed formula is not satisfied in both models.*

3. *$\| CC_{i\to j}(\psi,\varphi) \|_{\mathbb{K}_M} = M$ iff $K_T \models \phi_1$ and $K_F \not\models \phi_1$, means the satisfaction degree of the formula in $\mathbb{K}_M$ is uncertain iff the transformed formula is satisfied in K_T and not satisfied in K_f.*

Before we start the proof, let us build the truth table from the algorithm. Figure 4.4 shows this truth table as built from Algorithm 2 (lines 34-42). It indicates that if K_T (with M=T) gives true, we take the corresponding join-irreducible element M. If it gives false, we ignore this element. Similarly, if K_F (with only T=T) gives true, we take the corresponding join-irreducible element T. If it gives false, we ignore this element. The final result is determined by taking the join of the individual results.

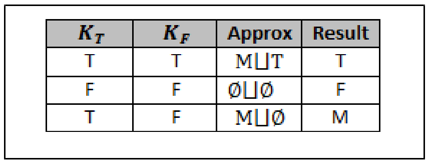
Proof. The proof of the theorem is straightforward, using induction with respect to the formula. Let's prove the theorem's statements one by one:

1. According to the truth table (Figure 4.4), if ϕ_1 is satisfied in K_T and K_F, then ϕ (the original formula which represents the commitment formula $CC_{i\to j}(\psi,\varphi)$) will be assigned the truth value "T" in $\mathbb{K}_M$. This proves the first point.

2. According to the same truth table, if ϕ_1 is not satisfied in K_T and K_F, then ϕ (the original commitment formula) will not be satisfied by taking the truth value "false" in $\mathbb{K}_M$. This proves the second point.

3. Finally, if ϕ_1 is satisfied in K_T and not satisfied in K_F, then as shown by the truth table, ϕ will be assigned the truth value "Maybe". This proves the third point.

$\square$



K_T	K_F	Approx	Result
T	T	M⊔T	T
F	F	∅⊔∅	F
T	F	M⊔∅	M

Figure 4.4: The truth table from Algorithm 2

4.6.3 Reduction Algorithm from 4v-CTL^{cc} to CTL^{cc}

Algorithm 3 generates two two-valued CTL^{cc} models K_T and K_F from the 4v-CTL^{cc} model $\mathbb{H}_c$. The lines from 1 to 6 function as in Algorithm 2. Here, R_T and AR$_T$ represent the transitions and commitment accessibility relations (explained in Example 1) in K_T, respectively. R_F and AR$_F$ represent the transitions and commitment accessibility relations in K_F, respectively. Lines 7 and 8 give the same state variables with their truth values in $\mathbb{H}_c$ to two sets S_T and S_F. Lines 9 and 10 build the set of states S_T considering every variable with the values TF or TT as True and the ones with the values FT and FF as False. Line 11 uses the function (get) to obtain the set S_T that includes all the states in this model. Lines 12 and 13 build the set of states S_F considering every variable with the values FT or TT as True and the ones with TF and FF as False. Line 14 gets the set S_F. Line 15 calls the procedure TRANSFORM from Algorithm 2 with the parameters $(S_T, S_F, \mathbb{H}_c)$ to establish the commitment and transition relations between states in S_T and S_F as in $\mathbb{H}_c$ in order to get the complete two-valued models K_T and K_F. Then, in lines 16 and 17, the algorithm calls the model checker MCMAS+ on each model with the input formula and saves the result in x and y. Finally, based on the results in line 18, we apply the approximations in lines 19-22 to get the final results.

Algorithm 3 Transform $\mathbb{H}_c = (S_c, R_c, \{\sim_{i\to j} | \ (i,j) \in A^2\}, I_T, \mathbb{O}_T)$ into into two-valued $K_T = (S_T, R_T, \{\sim_{i\to j} | \ (i,j) \in A^2\}, I_T, L_T)$ and two-valued $K_F = (S_F, R_F, \{\sim_{i\to j} | \ (i,j) \in A^2\}, I_F, L_F)$

```
1: **Input** the mv-model $\mathbb{H}_c$ and formula $\phi = CC_{i\to j}(\psi, \varphi)$
2: **Output** the two-valued models $K_T$ and $K_F$
3: $S_T = S_c$ and $S_F = S_c$
4: $I_T = I_c$ and $I_F = I_c$
5: Initialize $R_T = \emptyset$ and $R_F = \emptyset$
6: Initialize $\mathrm{AR}_T = \emptyset$ and $\mathrm{AR}_F = \emptyset$
7: Initialize $(V_T(S_T))(x) = (\mathbb{O}_c(S_c)(x)$
8: Initialize $(V_F(S_F))(x) = (\mathbb{O}_c(S_c)(x)$
9: For every $(V_T(S_T))(x) = TF \vee TT \Rightarrow (V_T(S_T))(x) = T$
10: For every $(V_T(S_T))(x) = FT \vee FF \Rightarrow (V_T(S_T))(x) = F$
11: **Get** $S_T$
12: For every $(V_F(S_F))(x) = FT \vee TT \Rightarrow (V_F(S_F))(x) = T$
13: For every $(V_F(S_F))(x) = TF \vee FF \Rightarrow (V_F(S_F))(x) = F$
14: **Get** $S_F$
15: $(K_T, K_F) \leftarrow \textsc{Transform}(S_T, S_F, \mathbb{H}_c)$
16: **Call MCMAS+ on $K_T$ and $\phi$**
17: **Call MCMAS+ on $K_F$ and $\phi$**
18: $x :=$ MCMAS result on $K_T$ and $\phi$ and $y :=$ MCMAS result on $K_F$ and $\phi$
19: If $x =$ True and $y =$ True then, the final result: $TF \sqcup FT = TT$
20: If $x =$ False and $y =$True then, the final result: $\emptyset \sqcup FT = FT$
21: If $x =$True and $y =$ False then, the final result: $TF \sqcup \emptyset = TF$
22: If $x =$ False and $y =$ False then, the final result: $\emptyset \sqcup \emptyset = FF$
```

Example 8. In Figure 4.5, the four-valued commitment model ($\mathbb{H}_c$) consists of three states with transitions and commitment accessibility relations. The model is designed by two experts having different points of view (disagreement) about the system's behavior. This disagreement is captured using the truth values over the four-valued lattice L_4. The formula $CC_{i\to j}(\psi, \phi)$ shown in the figure is verified over this model after they transformed into two two-valued commitment models, K_T considers TF as T and FT as F, and K_F considers FT as T and TF as F. The reason it transformed into two models is that the number of transformed models depends on the number of the join-irreducible elements in the used lattice, which is L_4 in our case. The formula is verified by TF as there are conflicting viewpoints about ψ in state s_1. After model transformation, the commitment formula is verified by the result True in K_T (with TF true). This means we consider the element TF for the final result. The same formula is verified by the value Flase in K_F (FT true). Therefore we ignore the corresponding element FT for the final result and assign an empty set instead. The final result is TF. This gives us information that the first expert says yes, and the second says no about the same formula.

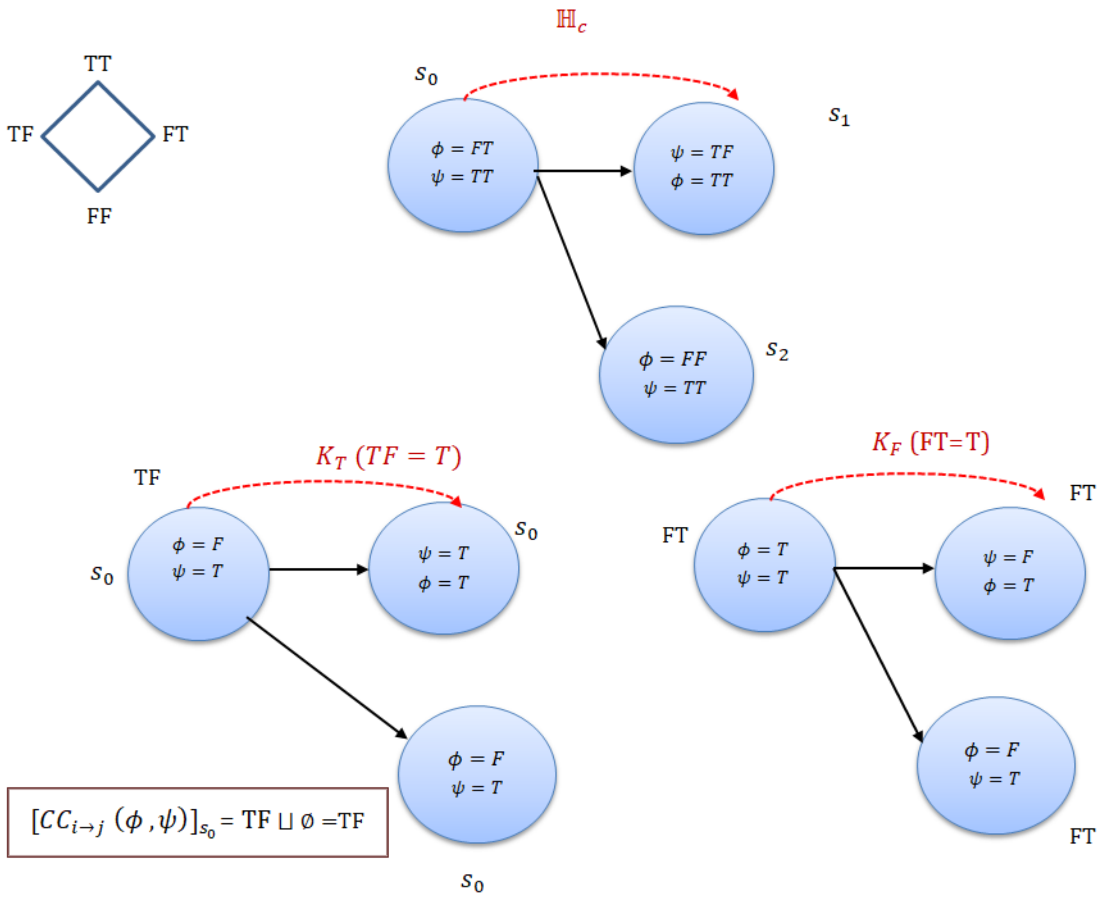


Figure 4.5: Transforming the 4v-model $\mathbb{U}_M$ into two two-valued models K_T and K_F

4.6.4 Soundness of Algorithm 3

Theorem 3. *(Soundness of Algorithm 3) Let $\mathbb{H}_c$ be a 4v-CTL^{cc} model and $\phi = CC_{i\to j}(\psi, \varphi)$ be a conditional commitment formula to be verified over this model. Also, let K_T and K_F be the corresponding two-valued CTL^{cc} models.*

We have:

1. *$\parallel \phi \parallel_{\mathbb{H}_c} = TT$ iff $K_T \models \phi$ and $K_F \models \phi$, means the satisfaction degree of the formula in the four-valued model $\mathbb{H}_c$ is TT (agreement) iff the formula ϕ is satisfied in both transformed models.*

2. *$\parallel \phi \parallel_{\mathbb{H}_c} = FF$ iff $K_T \not\models \phi$ and $K_F \not\models \phi$, means the satisfaction degree of the formula in $\mathbb{H}_c$ is FF (agreement) iff the formula ϕ is not satisfied in both transformed models.*

3. *$\parallel \phi \parallel_{\mathbb{H}_c} = TF$ iff $K_T \models \phi$ and $K_F \not\models \phi$, means the satisfaction degree of the formula in $\mathbb{H}_c$ is TF (disagreement) iff ϕ is satisfied in the first model and not satisfied in the second one.*

4. *$\parallel \phi \parallel_{\mathbb{H}_c} = FT$ iff $K_T \not\models \phi$ and $K_F \models \phi$, means the satisfaction degree of the formula in $\mathbb{H}_c$ is FT (disagreement) iff ϕ is not satisfied in the first model and satisfied in the second one.*

To prove this theorem, we first build a truth table from the algorithm. Figure 4.6 shows this truth table as built from Algorithm 3 (lines 18-22). It says if K_T (with TF=T) gives true, we take the corresponding join-irreducible element TF. If it is false, then we ignore this element. Similarly, if K_F (with FT=T) gives true, we take the corresponding join-irreducible element FT. If it is false, then we ignore this element. The final result is determined by taking the join of the individual results.

Proof. The proof of this theorem is straightforward using induction. Let's prove these statements one by one:

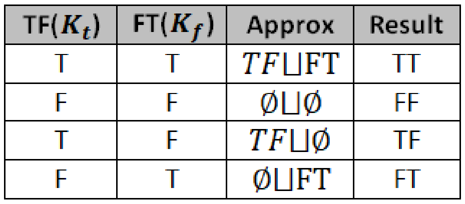
1. According to the truth table (Figure 4.6), if ϕ is satisfied in K_T and K_F then ϕ will be assigned the truth value "TT" in $\mathbb{H}_c$. This proves the first point.

2. According to the same truth table, if ϕ is not satisfied in K_T and K_F, then ϕ will be assigned the truth value "FF" in $\mathbb{H}_c$. This proves the second point.

3. Finally, if ϕ is satisfied in K_T and is not satisfied in K_F, then ϕ will be assigned the truth value "TF". This proves the third point.

4. If ϕ is not satisfied in K_T and satisfied in K_F, then ϕ will be assigned the truth value "FT". This proves the fourth point.

$\square$



TF(K_t)	FT(K_f)	Approx	Result
T	T	$TF \sqcup$FT	TT
F	F	$\emptyset \sqcup \emptyset$	FF
T	F	$TF \sqcup \emptyset$	TF
F	T	$\emptyset \sqcup$FT	FT

Figure 4.6: The truth table from Algorithm 3

4.6.5 Reduction Algorithm from 3v-CTL^c to CTL

Since Algorithm 2 is applicable to transform 3v-CTL^c to CTL^c, we produce a new reduction algorithm for reducing the problem of model checking 3v-CTL^c to CTL using our new tool **NuSMV-interactor** and compare the results obtained from the two algorithms. The algorithm depends on the join-irreducible elements in a given lattice. The general idea of this algorithm is to transform the 3v-CTL^c into two CTL^c, and then each model transforms to a CTL model. By doing so, we become able to use another efficient checker, which is NuSMV.

The complete procedure is shown in Algorithm 4 where the input is the three-valued model $\mathbb{U}_M$, and the formula ψ represents the commitment formula $C_{i\to j}(\varphi)$ in 3v-CTL^c. The outputs are the transformed two-valued models D_T and D_F and a CTL formula ϕ_1 defined as in Algorithm 1. Line 3 transforms the commitment formula to CTL via calling Procedure 2. Following the transformation strategy in [28], in this procedure, to preserve the CTL semantics, it is crucial to ensure that the transformation does not impact the temporal operators. Specifically, when introducing new states and transitions (that represent the commitment content) to the corresponding CTL models D_T and D_F, we must verify that the path through which the formula is satisfied in the original model $\mathbb{U}_M$ remains satisfied in the corresponding path of the translated models. Therefore, the fresh atomic proposition χ with negation is added to the transformed formula to show that the new state and relation added to present the commitment accessibility relation are not counted in the CTL model verification. The atomic proposition α^{ij} is introduced to represent the presence of an accessible state. Along each path, if the subsequent state on that path satisfies the atomic proposition α^{ij}, then the subsequent of this state also satisfies the transformed commitment formula ψ. After transforming the commitment formula to CTL, line 4 calls Procedure 1 to get the formula ϕ_1 as explained in Algorithm 1. Line 5 assigns the same states and initial states in $\mathbb{U}_M$ to each two-valued model. Lines from 6-14 function as lines 6-13 in Algorithm 1. Line 15 gets the complete sets of states for the two models. Line 16 calls Procedure 3 to build the relations of the two CTL models. In this procedure, every set of states is linked by the original relations and the new relations are added to present the commitment content on the CTL model. More specifically, the algorithm checks if there is a commitment accessibility relation between two states in $\mathbb{U}_M$, then an intermediate state will be added with two fresh atomic propositions (α^{ij}) and (χ) where the first represents the commitment accessibility relation between two intelligent agents i and j. The second is to know that the intermediate state is new and not included in the original commitment model. The same line (16) returns the transformed models D_T and D_F. Then, From lines 17 to 27, the algorithm calls NuSMV over these models with the formula ϕ_1 and gets the final results as explained in Algorithm 1.

Algorithm 4 Transform $\mathbb{U}_M = (S_M, R_M, \{\sim_{i\to j} \mid (i,j) \in A^2\}, I_M, \mathbb{O}_M)$ into $D_T = (S_T, R_T, I_T, V_T)$ and $D_F = (S_F, R_F, I_F, V_F)$

```
 1: **Input** the mv- model $\mathbb{U}_M$ and formula $\phi = C_{i\to j}(\varphi)$
 2: **Output** the CTL models $D_T, D_F$ and $\phi_1$
 3: $\phi' \leftarrow$ TRANSFORCOMMMULA($\phi$)
 4: $\phi_1 \leftarrow$ TRANSFORMULA($\phi'$)
 5: $(S_T = S_M$ and $S_F = S_M)$ and $(I_T = I_M$ and $I_F = I_M)$
 6: Initialize $R_T = \emptyset$ and Initialize $R_F = \emptyset$
 7: Initialize $(L_T(S_T))(x) = (\mathbb{O}_M(S_M))(x)$
 8: For every $(L_T(S_T))(x) = M \Rightarrow$ replace $(L_T(S_T))(x)$ by $(L_T(S_T))(x) = T$ and add
    $(L_T(S_T))(x-) = T$
 9: For every $(L_T(S_T))(x) = T \Rightarrow$ add $(L_T(S_T))(x-) = F$
10: For every $(L_T(S_T))(x) = F \Rightarrow$ add $(L_T(S_T))(x-) = T$
11: Initialize $(L_F(S_F))(x) = (\mathbb{O}_M(S_M))(x)$
12: For every $(L_F(S_F))(x) = M \Rightarrow$ replace $(L_F(S_F))(x)$ by $(L_F(S_F))(x) = F$ and add
    $(L_F(S_F))(x-) = F$
13: For every $(L_T(S_T))(x) = T \Rightarrow$ add $(L_T(S_T))(x-) = F$
14: For every $(L_T(S_T))(x) = F \Rightarrow$ add $(L_T(S_T))(x-) = T$
15: **Get** sets $S_T$ and $S_F$
16: $(D_T, D_F) \leftarrow$ TranfToCTL($S_T, S_F, \mathbb{U}_M$)
17: **Call NuSMV on** $D_T$
18: **if** $D_T \not\models \phi_1$ **then**
19:     "Formula is False"
20: **else**
21:     **Call NuSMV on** $D_F$
22:     **if** $D_F \models \phi_1$ **then**
23:         "Formula is True"
24:     **else**
25:         "Formula is uncertain (M)"
26:     **end if**
27: **end if**
```

Procedure 2 TRANSFORM CTL^c FORMULA ψ INTO CTL FORMULA $f(\psi)$

```
1: procedure TRANSFORCOMMMULA($\psi$)
2:     $f(x) = x$ if $x \in AP$
3:     $f(\neg\psi) = \neg f(\psi)$
4:     $f(\phi \vee \psi) = f(\phi) \vee f(\psi)$
5:     $f(EX\psi) = EXf(\phi) \wedge \neg\chi)$
6:     $f(E(\phi \cup \psi)) = E((f(\phi) \wedge \neg\chi) \cup (f(\psi) \wedge \neg\chi))$
7:     $f(EG\psi) = EG(f(\phi) \wedge \neg\chi)$
8:     $f(C_{i\to j}(\psi)) = AX(\alpha^{ij} \to AXf(\psi))$
9:     Return: $f(\psi)$
10: end procedure
```

Procedure 3 Build the state's relations of the CTL models

```
procedure TranfToCTL($S_T, S_F, \mathbb{U}_M$)
    for each $(s_M, s^{'}_M) \in S^2_M$ do
        if $(s_M, s^{'}_M) \in R_M$ then
            $R_T := R_T \cup \{(s_M, s^{'}_M)\}$
            if $s_M \sim_{i \to j} s^{'}_M$ for all $(i, j) \in A^2$ then
                if $\exists s^{''}$ such that $((s, s^{''}), (s^{''}, s^{'}) \in R_T$ and $\chi \in V_T(s^{''})$ then
                    $V_T(s^{''}) := V_T(s^{''}) \cup \{\alpha^{ij}\}$
                else
                    $S_T := S_T \cup \{s^{''}\}$
                    $R_T := R_T \cup \{(s, s^{''}), (s^{''}, s^{'})\}$ and $V_T(s^{''}) := \{\chi, \alpha\}$
                end if
            end if
        end if
    end for
    return $t$
    for each $(s_M, s^{'}_M) \in S^2_M$ do
        if $(s_M, s^{'}_M) \in R_M$ then
            $R_F := R_F \cup \{(s_M, s^{'}_M)\}$
            if $s_M \sim_{i \to j} s^{'}_M$ for all $(i, j) \in A^2$ then
                if $\exists s^{''}_M$ such that $((s_M, s^{''}_M), (s^{''}, s^{'}) \in R_F$ and $\chi \in V_F(s^{''})$ then
                    $V_F(s^{''}) := V_F(s^{''}) \cup \{\alpha^{ij}\}$
                else
                    $S_F := S_F \cup \{s^{''}\}$
                    $R_F := R_F \cup \{(s, s^{''}), (s^{''}, s^{'})\}$ and $V_F(s^{''}) := \{\chi, \alpha\}$
                end if
            end if
        end if
    end for
    return $(D_T, D_F)$
end procedure
```

Example 9. The algorithm is exemplified in Figure 4.7. In this figure, starting with the 3v-CTL^c model of unconditional commitment ($\mathbb{U}_M$), the algorithm takes this model and transforms it into two two-valued CTL^c models (here we include only the positive model, which considers M as true) and then the obtained models are transformed into two CTL models D_T and D_F. To simplify the example, we include only D_T. The CTL^c model is included implicitly in the algorithm. As shown in the figure, the model D_T includes a new state s_{01} added between s_0 and the accessible state s_1. This state represents the commitment content in this CTL model captured by α^{ij}. In addition, a fresh atomic proposition χ is added to this state to capture the fact that the state does not exist in the original commitment model. By doing so, a CTL-based model checker such as NuSMV can deal with a CTL system with the presence of commitment.

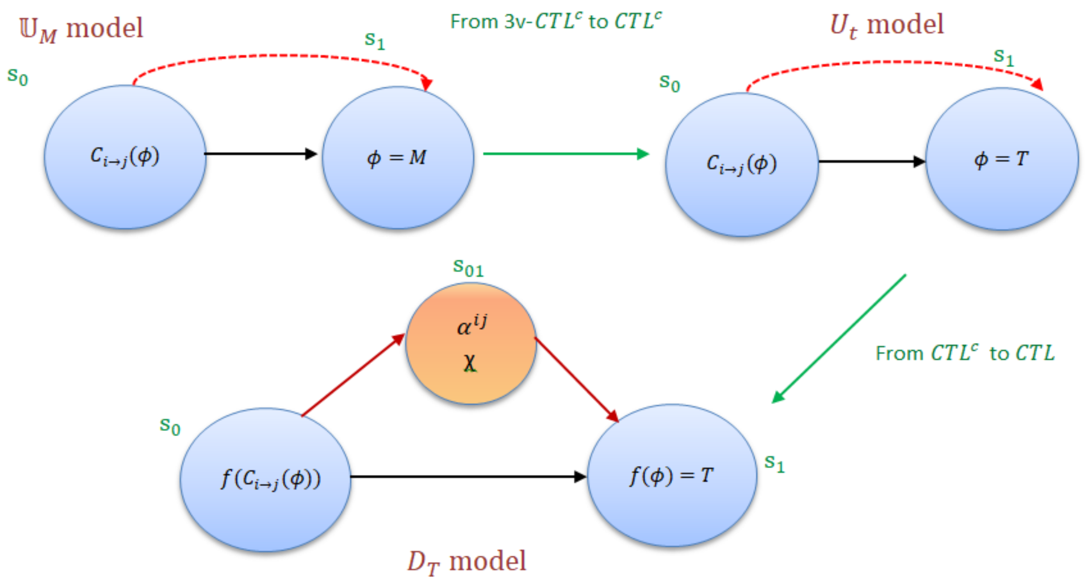


Figure 4.7: Transformation example from 3v-CTL^c to CTL (only the models consider M as true).

4.6.6 Soundness of Algorithm 4

Theorem 4. *(Soundness of Algorithm 4) Let $\mathbb{U}_M$ be a 3v-CTL^c model and $C_{i\to j}(\varphi)$ be an unconditional commitment formula to be verified over this model. Also, let D_T, D_F and ϕ_1 be the corresponding two-valued models and formulae in CTL.*

We have:

1. *$\parallel C_{i\to j}(\varphi) \parallel_{\mathbb{U}_M} = T$ iff $D_T \models \phi_1$ and $D_F \models \phi_1$, means the satisfaction degree of the commitment formula in the three-valued model $\mathbb{U}_M$ is true iff the CTL formula ϕ_1 is true in both CTL models.*

2. *$\parallel C_{i\to j}(\varphi) \parallel_{\mathbb{U}_M} = F$ iff $D_T \not\models \phi_1$ and $D_F \not\models \phi_1$, means the satisfaction degree of the formula in $\mathbb{U}_M$ is false iff the CTL formula ϕ_1 is false in both CTL models.*

3. $\parallel C_{i \to j}(\varphi) \parallel_{\mathbb{U}_M} = M$ *iff* $D_T \models \phi_1$ *and* $D_F \not\models \phi_1$*, means the satisfaction degree of the formula in* $\mathbb{U}_M$ *is uncertain iff the CTL formula* ϕ_1 *is true in CTL model* D_f *and false in* D_T*.*

Proof. The proof of this theorem is similar to the proof of Theorem 1.

1. According to the truth table (Figure 3.6, (a)) that can be built from Algorithm 4 (lines 18-27), if ϕ_1 is satisfied in D_T and D_F then ϕ (the original formula) will be satisfied by truth value "true" in $\mathbb{U}_M$. This proves the first point.

2. According to the same truth table, if ϕ_1 is not satisfied in D_T and D_F then ϕ (the original formula) will not be satisfied by taking the truth value "false" in $\mathbb{U}_M$. This proves the second point.

3. Finally, if ϕ_1 is satisfied in D_T and didn't satisfy in D_F, then the original formula ϕ will be verified by truth value "Maybe". This proves the third point.

□

4.6.7 Reducing 4v-CTL^c to CTL

We introduce Algorithm 5 to reduce 4v-CTL^c to CTL. The algorithm inputs a 4v-CTL^c model ($\mathbb{Y}_c$) and the commitment formula and outputs two CTL models in lines 1 and 2. Line 3 calls Procedure 2 to transform the commitment formula into CTL. Line 4 assigns two sets of states S_T and S_F with the same number of states in $\mathbb{Y}_c$. Line 5 assigns the same initial states as well. Line 6 initializes the relations between states by empty sets. Line 7 gives the same state variables with their truth values in $\mathbb{Y}_c$ to two sets S_T and S_F. Line 8 considers the variables of the states in S_T with truth values TT and TF as T, and line 9 considers the ones with FF and FT as F. The opposite applies to lines 10 and 11 for the set S_F. Line 12 gets the sets S_T and S_F containing the states with the changed truth values variables and without transition relations. Line 13 calls procedure 3 with the parameters (S_T, S_F, $\mathbb{Y}_c$) to establish the new transition relations between states in S_T and S_F in order to get the complete CTL models D_T and D_F. Lines 14 and 15 call the NuSMV over each

CTL model and the CTL formula. Line 16 saves the results in the variables z and y to apply the comparisons and approximations in 4.6.3 as the underlying lattice in this algorithm is the four-valued lattice.

Algorithm 5 Transform $\mathbb{Y}_c = (S_c, R_c, \{\sim_{i\to j} \mid (i,j) \in A^2\}, I_c, \mathbb{O}_c)$ into $D_T = (S_T, R_T, I_T, L_T)$ and $D_F = (S_F, R_F, I_F, L_F)$

```
1: Input the 4v- model $\mathbb{Y}_c$ and formula $\phi = C_{i\to j}(\varphi)$
2: Output the CTL models $D_T, D_F$ and $\phi_1$
3: $\phi_1 \leftarrow$ TRANSFORCOMMMULA($\phi$)
4: $S_T = S_c$ and $S_F = S_c$
5: $I_T = I_c$ and $I_F = I_c$
6: Initialize $R_T = \emptyset$ and $R_F = \emptyset$
7: Initialize $(L_T(S_T))(x) = (\mathbb{O}_c(S_c)(x)$ and $(L_F(S_F))(x) = (\mathbb{O}_c(S_c)(x)$
8: $(L_T(S_T))(x) = TF \vee TT \Rightarrow (L_T(S_T))(x) = T$ for each $s_T \in S_T$ and $s_c \in S_T$ such that
   $s_T = s_c$
9: $(L_T(S_T))(x) = FT \vee FF \Rightarrow (L_T(S_T))(x) = F$ for each $s_T \in S_T$ and $s_c \in S_c$ such that
   $s_T = s_c$
10: $(L_F(S_F))(x) = FT \vee TT \Rightarrow (L_F(S_F))(x) = T$ for each $s_F \in S_F$ and $s_{\in}S_c$ such that
    $s_F = s_c$
11: $(L_F(S_F))(x) = TF \vee FF \Rightarrow (L_F(S_F))(x) = F$ for each $s_F \in S_F$ and $s_T \in S_c$ such that
    $s_F = s_c$
12: Get sets $S_T$ and $S_F$
13: $(D_T, D_F) \leftarrow$ TranToCTL($S_T, S_F, \mathbb{Y}_c$)
14: Call NuSMV on $D_T$ and $\phi_1$
15: Call NuSMV on $D_F$ and $\phi_1$
16: $z :=$ NuSMV result on $D_T$ and $\phi_1$ and $y :=$ NuSMV result on $D_F$ and $\phi_1$
17: If $z =$ True and $y =$ True then, the final result: $TF \sqcup FT = TT$
18: If $z =$ False and $y =$True then, the final result: $\emptyset \sqcup FT = FT$
19: If $z =$True and $y =$ False then, the final result: $TF \sqcup \emptyset = TF$
20: If $z =$ False and $y =$ False then, the final result: $\emptyset \sqcup \emptyset = FF$
```

4.6.8 Soundness of Algorithm 5

Theorem 5. *(Soundness of Algorithm 5) Let $\mathbb{Y}_c$ and $C_{i\to j}(\varphi)$ be a 4v-CTL^c model and an unconditional commitment formula to be verified over this model. Also, let D_T, D_F and ϕ_1 be the corresponding two-valued models and formulae in CTL.*

We have:

1. *$\parallel C_{i\to j}(\varphi) \parallel_{\mathbb{Y}_c} = TT$ iff $D_T \models \phi_1$ and $D_F \models \phi_1$, means the satisfaction degree of the formula in the three-valued model $\mathbb{Y}_c$ is (true, true) if the CTL formula ϕ_1 is true in*

both CTL models.

2. $\parallel C_{i\to j}(\varphi)\parallel_{\mathbb{Y}_c}= FF$ *iff* $D_T \not\models \phi_1$ *and* $D_F \not\models \phi_1$*, means the satisfaction degree of the formula in* $\mathbb{Y}_c$ *is (false,false) if the CTL formula* ϕ_1 *is false in both CTL models.*

3. $\parallel C_{i\to j}(\varphi)\parallel_{\mathbb{Y}_c}= TF$ *iff* $D_T \models \phi_1$ *and* $D_F \not\models \phi_1$*, means the satisfaction degree of the formula in* $\mathbb{Y}_c$ *is (true, false) if the CTL formula* ϕ_1 *is true in CTL model* D_t *and false in* D_f*.*

4. $\parallel C_{i\to j}(\varphi)\parallel_{\mathbb{Y}_c}= FT$ *iff* $D_T \not\models \phi_1$ *and* $D_F \models \phi_1$*, means the satisfaction degree of the formula in* $\mathbb{U}_c$ *is (false, true) if the CTL formula* ϕ_1 *is false in CTL model* D_T *and true in* D_f*.*

Proof. Similar to Algorithm 4, the transformation from 4v-CTL^c to CTL is done in two steps. The first is transforming 4v-CTL^c to CTL^c and then transforming to CTL. The proof of the soundness of transforming CTL^c to CTL can be derived from Theorem 4. Based on the truth table from Algorithm 3, which can be obtained from lines 17 to 20, the proof of this theorem is straightforward as in Algorithm 3. □

4.7 Computational Complexity Analysis

The reasons for delving into the computational complexity of the model checking problem encompass the following objectives: 1) Establish a formal rationale demonstrating the efficacy of the proposed approach. 2) Determine the computational resources necessary to handle all instances of the problem, including the most challenging scenarios. 3) Provide a lucid understanding of the genuine computational challenge underlying the problem. 4) Conduct a comparative assessment of various model checking techniques. The hierarchical relationship between the common complexity classes is determined by the subset relation $\subseteq$ as follows: L $\subseteq$ NL $\subseteq$ P $\subseteq$ NP $\subseteq$ PSPACE $\subseteq$ NPSPACE $\subseteq$ EXPTIME $\subseteq$ EXPSPACE. These classes can be read from bottom to up as logarithmic space, nondeterministic logarithmic space, polynomial time, nondeterministic polynomial time, polynomial space, nondeterministic polynomial space, exponential time and exponential space. In this section, we provide the complexity analysis for our algorithms. As mentioned, our reduction algorithms basically transform the multi-valued model into two two-valued models. The two models exhibit similarities and solely differ in the truth values of atomic propositions. Because these transformations are similar and can be performed in parallel, it is enough to consider only one transformation for the complexity analysis. Therefore, our analysis centres on the model with uncertainty in which M is considered true.

The computational analysis of the algorithms that are handling models with inconsistency is performed following the same strategies applied in this section with models with uncertainty.

4.7.1 Time Complexity of Model Checking 3v-CTL^{cc} through Transformation to CTL^{cc}

In this subsection, we will prove that the problem of model checking 3v-CTL^{cc} through the transformation to CTL^{cc} is P-complete in time. Generally, a problem is classified as P-complete if it is solvable in polynomial time, and any problem in P can be reduced to it in polynomial time. The assessment of model checking complexity is contingent upon the scale of both the model and the representation of the formula employed. Consequently, it is imperative to elucidate the chosen input representation and state how we measure its size.

Proposition 1. *(Boundedness of atomic propositions). Let K_T be the positive CTL^{cc} model obtained from $\mathbb{K}_M$ in Algorithm 2. Moreover, let $Atom(K_T)$ be the number of atomic propositions in the model K_T and $Atom(\mathbb{K}_M)$ be the number of atomic propositions in model $\mathbb{K}_M$. We have: $Atom(K_T) = 2 \times Atom(\mathbb{K}_M)$*

Proof. The proof is straightforward from steps 7-9 in Algorithm 2. These steps introduce an additional atomic proposition in the model K_T for each atomic proposition. Thus, the number of atomic propositions in K_T, $Atom(K_T)$, is proportional to the number of atomic propositions in $\mathbb{K}_M$, $Atom(\mathbb{K}_M)$, with a scaling factor of 2. □

Proposition 2. *(Boundedness of model transformation). Let $|K_T|$ and $|\mathbb{K}_M|$ be the size of K_T and $\mathbb{K}_M$ respectively. $|K_T|$ is linear with $|\mathbb{K}_M|$.*

Proof. As shown in Algorithm 2, the model K_T has the same number of states and relations as the model $\mathbb{K}_M$. The results flow then from Proposition 1. $\square$

Theorem 6. *(Explicit 3v-CTL^{cc} model checking: upper bound). The model checking problem of 3v-CTL^{cc} through transformation to CTL^{cc} can be solved in time $\mathcal{O}(|\mathbb{K}_M| \times |\phi|)$. Where $|\mathbb{K}_M|$ and $|\phi|$ are the size of the model and length of the formula, respectively.*

Proof. 3v-CTL^{cc} extends CTL^{cc}. It is known from [31] that the model checking problem for CTL^{cc} is linear in the size of the model and the length of the formula. From Proposition 2, $|K_T| = 2|\mathbb{K}_M|$ considering atomic variables as part of the state space. Moreover, from Procedure 1, it is evident that the length of the transformed formula ϕ_1 equals the original formula ϕ where the procedure only replaces each negated proposition $\neg x$ by $x-$, so the result. $\square$

Theorem 7. *(Explicit 3v-CTL^{cc} model checking: completeness) The model checking problem of 3v-CTL^{cc} through transformation to CTL^{cc} is P-complete.*

Proof. Membership in P (i.e., upper bound) follows from Theorem 6. Hardness in P (i.e., lower bound) followed by a reduction from model checking CTL^{cc} proved to be P-complete in [31]. $\square$

4.7.2 Space Complexity of Model Checking 3v-CTL^{cc} through Transformation to CTL^{cc}

Here, we prove that the complexity of 3v-CTL^{cc} model checking, through the transformation to CTL^{cc}, for concurrent programs [55] is PSPACE-complete. This finding indicates that there exists an algorithm capable of solving the problem within polynomial space relative to the size of the components comprising concurrent programs and the length of the formula undergoing model checking.

Theorem 8. *(Polynomial reduction of 3v-CTL^{cc} model checking: upper bound) Let $\sqsubseteq_{psr}$ denote the polynomial-space reduction. The problem of mv-model checking 3v-CTL^{cc} can* *be reduced to the problem of model checking CTL^{cc} in a polynomial space. Formally, mv-MC(3v-CTL^{cc}) $\sqsubseteq_{psr}$ MC(CTL^{cc}).*

Proof. Transforming the 3v-CTL^{cc} model and a CTL^{cc} formula into the corresponding CTL^{cc} model and formula could be computed by a deterministic Turing Machine (TM) in space $O(log\ n)$ where n is the size of the input 3v-CTL^{cc} model and polynomial space with regard to the length of the input formula. TM performs one-by-one the following steps: (1) reads in the input tape a model of 3v-CTL^{cc}; (2) for each model, TM sequentially generates on the output tape the same states and the same valuations (mapping of atomic propositions); (3) replaces the value of any atomic proposition x equal to M ($x = M$) with T ($x = T$); (4) adds a fresh negotiation version for every atomic proposition in every state; (5) build the transitions between the obtained states where for the transitions $(s, s^{'})$ in the 3v-CTL^{cc} (input model), it writes the transitions in the set R_T. The same applies to the commitment accessibility relations (AR_T) in the input model. The performed writing operations are clearly logarithmic in space as this transformation is conducted on-the-fly and step-by-step. Furthermore, as shown in the proof of Theorem 6 any CTL^{cc} formula is transformable into a nun-negated CTL^{cc} formula where its length equals the length of the input formula. Based on this analysis, it is clear that all these transformations are polynomial spaces in the size of the input formula. □

Theorem 9. *3v-CTL^{cc} Model Checking for Concurrent Programs: Completeness). The space complexity of the 3v-CTL^{cc} model checking through transformation to CTL^{cc} for concurrent programs is PSPACE-complete with respect to both the size of the program components and the length of the formula.*

Proof. Given that model checking CTL^{cc} is proved PSPACE-complete for concurrent programs [31], it sets a lower bound for model checking 3v-CTL^{cc} at PSPACE. Notably, 3v-CTL^{cc} encompasses CTL^{cc} by incorporating both 3v-CTL modalities and commitment modalities. The upper bound within PSPACE is given by Theorem 8, thereby affirming the result. □

4.7.3 Time Complexity of Model Checking 3v-CTL^c through Transformation to CTL

In this subsection, we will prove that model checking 3v-CTL^c through transformation to CTL is also P-complete. We transform the 3v-CTL^c logic into CTL in two steps. The first is from 3v-CTL^c into CTL^c, and this is addressed in the previous subsection. The second is from CTL^c into CTL following the same strategy as in [24] and in[31] excluding the condition ψ in the commitment formula.

Proposition 3. *(Boundedness of model transformation). Let D_T be a positive CTL model obtained from the 3v-CTL^c model $\mathbb{U}_M$ in Algorithm 4. Let $|D_T|$ and $|\mathbb{U}_M|$ be the size of D_T and $\mathbb{U}_M$, respectively. $|D_T| < 3|\mathbb{U}_M|$. We have $|D_T| \leq 3|\mathbb{U}_M|$.*

Proof. Let $|A_M|$ be the number of commitment accessibility relations in $\mathbb{U}_M$. We have $|\mathbb{U}_M| = |R_M| + |S_M| + |A_M|$ and $|D_T| = |S_T| + |R_T|$. As explained, each accessibility relation is translated into one additional state and two transitions in D_T. Based on this, and using the same argument as in Proposition 1 (considering the atomic propositions part of the state space), we obtain $|D_T| \leq |R_M| + 2|S_M| + 3|A_M|$. In a typical scenario, multiple reachable states could exist between any given pair of states. Indeed, the other accessibility relations can be established by the already added states and transitions, which gives the result: $|D_T| \leq 3(|R_M| + |S_M| + |A_M|)$. $\square$

Theorem 10. *(Explicit 3v-CTL^{cc} model checking: upper bound) The 3v-CTL^c model checking problem can be solved in in time $\mathcal{O}(|\mathbb{U}_M| \times |\phi|)$. Where $|\mathbb{U}_M|$ is the size of the model, and $|\phi|$ is the length of the formula.*

Proof. 3v-CTL^c can be reduced to CTL. From [43], it is known that the model checking problem for CTL is linear in the size of the model and the length of the formula. From Proposition 3, $|D_T| \leq 3|\mathbb{U}_M|$. The result follows from the fact that the length of the transformed CTL formula ϕ_1 from Algorithm 4 is linear with the length of the input commitment formula ϕ as proved in Proposition 2 and Theorem 4 in [24]. $\square$

Theorem 11. *(Explicit 3v-CTL^c model checking: completeness). The model checking problem of 3v-CTL^c through transformation to CTL is P-complete.*

Proof. Membership in P (i.e., upper bound) follows from Theorem 10. Hardness in P (i.e., lower bound) follows from the P-completeness of explicit model checking CTL [10]. □

4.7.4 Space Complexity of of Model Checking 3v-CTL^c through Transformation to CTL

We prove in this subsection that the complexity of 3v-CTL^c model checking through the transformation to CTL for concurrent programs is also PSPACE-complete.

Theorem 12. *(Polynomial reduction of 3v-CTL^c model checking: upper bound) Let $\sqsubseteq_{psr}$ denote the polynomial-space reduction. The problem of mv-model checking 3v-CTL^c can be reduced to the problem of model checking CTL in a polynomial space. Formally, mv-MC(3v-CTL^c) $\sqsubseteq_{psr}$ MC(CTL).*

Proof. The transformation of the 3v-CTL^c model and CTL^c formula into the corresponding CTL model and formula could be computed by a deterministic Turing Machine (TM) in space $O(log\ n)$, where n is the size of the input 3v-CTL^c model, and polynomial space concerning the length of the CTL^c formula. Regarding the two transformed CTL models, for D_T, TM performs the following one-by-one steps: (1) reads in the input tape a model of 3v-CTL^c; (2) generates in the output tape the same states with the same atomic propositions; (3) replaces the value of any atomic proposition x equal to M ($x = M$) with T ($x = T$); (4) adds a fresh negotiation version for every atomic proposition in every state; (5) build the transitions between the obtained states (R_T). For the commitment accessibility relations $\sim_{i\to j}$, TM reads the relations between two given states in the input model, and for each one, it adds an intermediate state to the set of states (S_T). Each intermediate state is labeled with two fresh propositional variables: 1) α^{ij} presents the commitment accessibility relation between agents i and j, and 2) χ , indicates that the two transitions with the corresponding state do not already exist in the original model (3v-CTL^c). All of the performed writing operations exhibit a clear logarithmic space complexity due to the on-the-fly, step-by-step nature of the transformation. Additionally, according to Proposition 2 in [24], it is demonstrated that any CTL^c formula can be converted into a CTL formula with a length that is linearly bounded by that of the original input formula. These recursive transformations are evidently polynomial in space concerning the length of the input formula, thus establishing the theorem. □

Theorem 13. *3v-CTL^c Model Checking for Concurrent Programs: Completeness). The space complexity of the 3v-CTL^c model checking through transformation to CTL for concurrent programs is PSPACE-complete with respect to both the size of the program components and the length of the formula.*

Proof. Given that model checking CTL is PSPACE-complete for concurrent programs [43], it sets a lower bound for model checking 3v-CTL^c at PSPACE-complete. Notably, 3v-CTL^c encompasses CTL through CTL^c by incorporating both 3v-CTL modalities and commitment modalities. The upper bound within PSPACE is supported by Theorem 12, thereby affirming the result. □

The problem of mv-model checking 4v-CTL^c, 4v-CTL^{cc} and 3v-CTL^{cc} is decided to be P-complete as the proofs can be obtained following the same strategies applied in this section.

Chapter 5

Modeling and Verifying IoT/IS Systems with mv-Trust Logics

5.1 Overview and Motivation

Expanding on the strategies employed to broaden the scope of commitment logic, we likewise extend the ambit of trust logic to encompass its multi-valued versions. Trust logic, denoted as TCTL, constitutes a natural expansion of the well-established CTL, integrating trust modalities while introducing novel syntax and semantics [21]. Subsequently, we further refine this framework into the multi-valued TCTL (mv-TCTL).

This newly formulated logic segregates into two distinct branches: firstly, the three-valued TCTL (3v-TCTL), tailored for addressing uncertainty within IoT/IS systems; secondly, the four-valued TCTL (4v-TCTL), geared towards handling inconsistencies within the same systems. To validate the effectiveness of these developments, we applied these logics to model various IoT and IS systems.

In tandem with these advancements, we have devised two transformation algorithms capable of converting mv-TCTL into both TCTL and CTL. Crucially, we employ our state-of-the-art model checker, **MV-Checker**, to seamlessly apply these transformation algorithms. This facilitates a rigorous and automated verification process as we engage with MCMAS^t for comprehensive analysis.

5.2 Modelling Uncertainty over IS/IoT with mv-Trust)

5.2.1 3v-TCTL

In this section, we introduce our three-valued logic of trust named 3v-TCTL [5]. This logic is used to reason about uncertainty over Iot/IS systems with trust protocols. In particular, 3v-TCTL is an extension to the multi-valued CTL (mv-CTL) presented in section 2.6 by adding the trust modalities to this logic. As mentioned, the multi-valued logic can be used for reasoning about uncertainty when the used underlying lattice is the three-valued lattice (L_3). In the 3v-TCTL, the lattice L_3 is the base of this logic, where the third truth value, "M", is used to capture the missing or uncertain information in trust models. For example, when we need to verify whether the trust formula $T(i, j, \phi)$ holds in the system and at the same time we miss information about ϕ, we can capture this uncertainty by giving the value "M" to ϕ.

5.2.2 Model of 3v-TCTL

The 3v-TCTL model expressed by ($\mathbb{T}_M$) is an extension of the TCTL model explained in 2.4 where the lattice structure ($L_3, \sqcup, \sqcap$) is used for extending this model. We replaced the valuation function Fn by $\mathbb{O}$ defined in mv-CTL.

5.2.3 Syntax

The 3v-TCTL is essentially the same as TCTL in terms of syntax, except that formulas are evaluated over lattice L_3.

5.2.4 Semantics

The semantics of this logic is based on the three-valued case of mv-CTL. This semantic relies mainly on the concept of *multi-valued sets* and *multi-valued relations* that are explained in detail in [17, 2, 3]. Bellow is the extension of mv-CTL semantic by the three-valued semantic of the 3v-TCTL.

- $\parallel T(i,j,\phi) \parallel_{\mathbb{T}_M} (s) = T$ *iff* $s \not\models \phi$ *and* $\forall s^{'} \neq s$ such that $s \sim_{i\rightarrow j} s^{'}$, we have $\parallel \varphi \parallel_{\mathbb{T}_M} (s^{'}) = T$.

  According to this semantics, the satisfaction degree of the trust formula $T(i,j,\phi)$ in state s of the model $\mathbb{T}_M$ is evaluated to "true" if and only if in all the accessible states $s^{'}$ the truth degree of φ is T.

- $\parallel T(i,j,\phi) \parallel_{\mathbb{T}_M} (s) = M$ *iffs* $\not\models \phi$ *and* $\forall s^{'} \neq s$ such that $s \sim_{i\rightarrow j} s^{'}$,we have $\parallel \varphi \parallel_{\mathbb{T}_M} (s^{'}) \neq F$ and $\exists s^{'} \in S$ such that $s \sim_{i\rightarrow j} s^{'}$ and $\parallel \varphi \parallel_{\mathbb{T}_M} (s^{'}) = M$.

  According to this semantics, the satisfaction degree of the trust formula $T(i,j,\phi)$ in the system state s of $\mathbb{T}_M$ is M if and only if in all the states $s^{'}$ that are accessible, the truth degree of the content ϕ is not equal to F and there exists at least one accessible state $s^{'}$ where ϕ is evaluated to M.

5.3 Modelling Inconsistency over IoT/IS with mv-Trust)

5.3.1 4v-TCTL

This section introduces the new four-valued trust logic named (4v-TCTL) [4]. This logic is used to reason about inconsistency over IoT/IS with trust protocols. This logic is an extension of the multi-valued CTL mv-CTL introduced in Section 2.6, where trust modalities are seamlessly integrated. The foundation of 4v-TCTL lies in the four-valued lattice (L_4). Within this lattice, four distinct truth values are defined: TT and FF signify unanimous agreement among designers regarding the satisfaction or dissatisfaction of a formula in a specific state, respectively. The values TF and FT, conversely, represent disagreement. Specifically, TF indicates that the first designer asserts "true" while the second asserts "false" concerning the satisfaction of a formula in a given state. Conversely, FT signifies that the first designer argues "false" while the second contends "true." To illustrate, consider the scenario where we seek to verify the trust formula $T(i,j,\phi)$ in the system while simultaneously encountering differing opinions from two designers regarding the satisfaction of ϕ. In this case, with the first designer asserting "true" and the second asserting "false," we encapsulate this disagreement by assigning the value "TF" to ϕ . This approach, grounded in 4v-TCTL and the associated L_4 lattice, offers a precise and robust method for handling inconsistencies within IoT/IS with trust, providing a foundation for effective reasoning and decision-making.

5.3.2 Model of 4v-TCTL

The 4v-TCTL model expressed by ($\mathbb{T}$) is an extension of the TCTL model explained in 2.4 where the lattice structure $(L_4, \sqcup, \sqcap)$ is used for extending this model. We replaced the valuation function Fn by $\mathbb{O}$ defined in mv-CTL.

5.3.3 Syntax

The 4v-TCTL is essentially the same as TCTL in terms of syntax, except that formulas are evaluated over lattice L_4.

5.3.4 Semantics

Consider $\mathbb{T}$ is a 4v-TCTL model and $T(i, j, \varphi)$ is a trusted formula to be verified over this model then:

- $\parallel T(i,j,\varphi) \parallel_{\mathbb{T}} (s) = TT$ iff $s \not\models \phi$ *and* $\forall s^{'} \in S$ *s.t.* $s \sim_{i \to j} s^{'}$ we have $\parallel \varphi \parallel_{\mathbb{T}} (s^{'}) = TT$.

  In this semantics, the degree to which the trust formula $T(i, j, \varphi)$ is satisfied in state s of the four-valued system $\mathbb{T}$ is TT when formula φ is not satisfied in the current state s, and in all accessible states s', the degree of satisfaction of formula φ is "true, true".

- $\parallel T(i,j,\varphi) \parallel_{\mathbb{T}} (s) = TF$ iff $s \not\models \phi$ *and* $\forall s^{'} \in S$ *s.t.* $s \sim_{i \to j} s^{'}$ we have $\parallel \varphi \parallel_{\mathbb{T}} (s^{'}) \neq FF$ and $\exists s^{'} \in S$ *s.t.* $s \sim_{i \to j} s^{'}$ and $\parallel \varphi \parallel_{\mathbb{T}} (s^{'}) = TF$.

  This implies that the trust formula's satisfaction degree is "TF" only when formula φ is not satisfied in the current state s, and the satisfaction degree of formula φ in all the accessible states $s^{'}$ is not "FF", at the same time, there exists at least one accessible state $s^{'}$ holds φ with the value "TF". In other words, the values of φ in the accessible states must include at least "TF" as "$TT \sqcap TF = TF$" and must not include "FF" as "$FF \sqcap TF \sqcap TT = FF$"

- $\parallel T(i,j,\varphi) \parallel_{\mathbb{T}} (s) = FT$ iff $s \not\models \phi \; and \, \forall s^{'} \in S \;\; s.t. \;\; s \sim_{i \to j} s^{'}$ we have $\parallel \varphi \parallel_{\mathbb{T}} (s^{'}) \neq FF$ and $\exists s^{'} \in S \;\; s.t. \;\; s \sim_{i \to j} s^{'}$ and $\parallel \varphi \parallel_{\mathbb{T}} (s^{'}) = FT$. This semantics implies that the trust formula's satisfaction degree is "FT" only when formula φ is not satisfied in the current state s, and the truth degree of φ in all accessible states s' is not equal to "FF", and there exists at least one accessible state s' in which φ holds with a value of "FT". Essentially, the accessible states must include at least "FT" as one of the possible truth values for φ, since "$TT \sqcap FT = FT$", while it should not include "FF", as "$FF \sqcap FT \sqcap TT = FF$".

5.4 Reduction-Based Multi-Valued Model Checking mv-TCTL

5.4.1 Reducing Algorithm of 3v-TCTL to TCTL

Algorithm 6 performs as Algorithm 2 except the input model and formula are a 3v-TCTL model and trust formula $T(i,j,\phi)$ in lines 1 and 2 and in lines 32 and 33 the tool interacts with MCMAS^t for trust.

Algorithm 6 Transform $\mathbb{T}_M = (S_M, R_M, \{\sim_{i\to j} | (i,j) \in A^2\}, I_M, \mathbb{O}_M)$ into two-valued $\mathcal{D}_T = (S_T, R_T, \{\sim_{i\to j} | (i,j) \in A^2\}, I_T, L_T)$ and two-valued $\mathcal{D}_F = (S_F, R_F, \{\sim_{i\to j} | (i,j) \in A^2\}, I_F, L_F)$

```
1: **Input** the mv-model $\mathbb{T}_M$ and a commitment formula $\phi = T(i,j,\varphi)$
2: $\phi_1 \leftarrow$ TRANSFORMULA($\phi$)
3: **Output** the two-valued trust models $\mathcal{D}_T$ and $\mathcal{D}_F$
4: $S_T = S_M$, $S_F = S_M$, $I_T = I_M$ and $I_F = I_M$
5: Initialize $R_T = \emptyset$, $R_F = \emptyset$, $AR_T = \emptyset$ and $\mathrm{AR}_F = \emptyset$
6: Initialize $(L_T(S_T))(x) = (\mathbb{O}_M(S_M))(x)$
7: For every $(L_T(S_T))(x) = M \Rightarrow$ replace $(L_T(S_T))(x)$ by $(L_T(S_T))(x) = T$ and add $(L_T(S_T))(x-) = T$
8: For every $(L_T(S_T))(x) = T \Rightarrow$ add $(L_T(S_T))(x-) = F$
9: For every $(L_T(S_T))(x) = F \Rightarrow$ add $(L_T(S_T))(x-) = T$
10: Initialize $(L_F(S_F))(x) = (\mathbb{O}_M(S_M))(x)$
11: For every $(L_F(S_F))(x) = M \Rightarrow$ replace $(L_F(S_F))(x)$ by $(L_F(S_F))(x) = F$ and add $(L_F(S_F))(x-) = F$
12: For every $(L_T(S_T))(x) = T \Rightarrow$ add $(L_T(S_T))(x-) = F$
13: For every $(L_T(S_T))(x) = F \Rightarrow$ add $(L_T(S_T))(x-) = T$
14: **Get** the sets $S_T$ and $S_F$
15: **procedure** TRANSFORM $(S_T, S_F, \mathbb{T}_M)$
16: **for each** $(s_M, s'_M) \in S_M^2$ **do**
17:     **if** $(s_M, s'_M) \in R_M$ **then**
18:         $R_T := R_T \cup \{(s_M, s'_M)\}$
19:         **if** $s_M \sim_{i\to j} s'_M$ **then** $\mathrm{AR}_T = \mathrm{AR}_T \cup \{s_M \sim_{i\to j} s'_M\}$
20:         **end if**
21:     **end if**
22: **end for**
23: **for each** $(s_M, s'_M) \in S_M^2$ **do**
24:     **if** $(s_M, s'_M) \in R_M$ **then**
25:         $R_F := R_F \cup \{(s_M, s'_M)\}$
26:         **if** $s_M \sim_{i\to j} s'_M$ **then** $\mathrm{AR}_F = \mathrm{AR}_F \cup \{s_M \sim_{i\to j} s'_M\}$
27:         **end if**
28:     **end if**
29: **end for**
30: **Return** $(\mathcal{D}_T, \mathcal{D}_F)$
31: **EndProcedure**
32: **Call MCMAS$^t$ on $\mathcal{D}_T$ and $\phi_1$**
33: **Call MCMAS$^t$ on $\mathcal{D}_F$ and $\phi_1$**
```

5.4.2 Soundness of Algorithm 6

Since the truth table of this algorithm is similar to the one obtained from Algorithm 2, the soundness proof of Algorithm 6 is straightforward.

5.4.3 Reducing Algorithm of 3v-TCTL to CTL

Algorithm 7 transforms the 3v-TCTL into CTL in order to use NuSMV checker. In line 1, the algorithm inputs the 3v-TCTL model and a trust formula and performs the same steps of Algorithm 4. Figure 5.1 explains the reduction process from 3v-TCTL into CTL. In particular, the algorithm starts by inputting a 3v-TCTL model. In this model, we must verify whether state s_0 satisfies the trust formula $T(i, j, \phi)$. We assume it is uncertain if ϕ holds in s_1. The main idea is to generate two TCTL models from the 3v-TCTL model. The first considers M as true (T), and the other considers M as false (F). To simplify the figure, we only show the TCTL model that considers M true. The next step transforms the two obtained models to CTL following the algorithm recently presented in [24]. To do so, a new state is added to the CTL model (the state in color orange, s_{01}, in the figure). The reason for adding this state is that CTL doesn't include the modality of trust in its syntax and semantics, and as a sequence, NuSMV checker will not recognize the trust relations between agents in the model. More specifically, the state s_{01} in the middle of s_0 and s_1 presents the trust between agents i and j in state s_1 where the atomic proposition χ indicates that this state is not within the original model's states and α^{ij} presents the trust between i and j. If these two atomic propositions hold in a state, this means a trust relation exists in the next state. The trust formula is also transformed into CTL formula as follows: $f(T(i, j, \phi)) = AX(\alpha^{ij} \rightarrow AXf(\phi))$.

Algorithm 7 Transform $\mathbb{T}_M = (S_M, R_M, \{\sim_{i\to j} | \ (i,j) \in A^2\}, I_M, \mathbb{O}_M)$ into $D_T = (S_T, R_T, I_T, V_T)$ and $D_F = (S_F, R_F, I_F, V_F)$

```
1: Input the mv- model $\mathbb{T}_M$ and formula $\phi = T(i, j, \varphi)$
2: Output the CTL models $D_T, D_F$ and $\phi_1$
3: $\phi' \leftarrow$ TRANSFORCOMMMULA($\phi$)
4: $\phi_1 \leftarrow$ TRANSFORMULA($\phi'$)
5: ($S_T = S_M$ and $S_F = S_M$) and ($I_T = I_M$ and $I_F = I_M$)
6: Initialize $R_T = \emptyset$ and Initialize $R_F = \emptyset$
7: Initialize $(L_T(S_T))(x) = (\mathbb{O}_M(S_M))(x)$
8: For every $(L_T(S_T))(x) = M \Rightarrow$ replace $(L_T(S_T))(x)$ by $(L_T(S_T))(x) = T$ and add $(L_T(S_T))(x-) = T$
9: For every $(L_T(S_T))(x) = T \Rightarrow$ add $(L_T(S_T))(x-) = F$
10: For every $(L_T(S_T))(x) = F \Rightarrow$ add $(L_T(S_T))(x-) = T$
11: Initialize $(L_F(S_F))(x) = (\mathbb{O}_M(S_M))(x)$
12: For every $(L_F(S_F))(x) = M \Rightarrow$ replace $(L_F(S_F))(x)$ by $(L_F(S_F))(x) = F$ and add $(L_F(S_F))(x-) = F$
13: For every $(L_T(S_T))(x) = T \Rightarrow$ add $(L_T(S_T))(x-) = F$
14: For every $(L_T(S_T))(x) = F \Rightarrow$ add $(L_T(S_T))(x-) = T$
15: Get sets $S_T$ and $S_F$
16: $(D_T, D_F) \leftarrow$ TranfToCTL($S_T, S_F, \mathbb{T}_M$)
17: Call NuSMV on $D_T$
18: if $D_T \not\models \phi_1$ then
19:     “Formula is False”
20: else
21:     Call NuSMV on $D_F$
22:     if $D_F \models \phi_1$ then
23:         “Formula is True”
24:     else
25:         “Formula is uncertain (M)”
26:     end if
27: end if
```

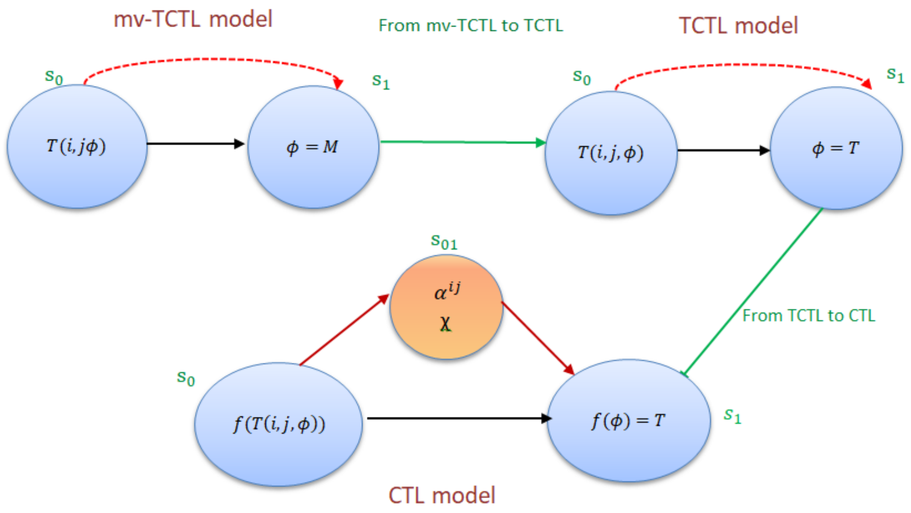


Figure 5.1: Transformation example from mv-TCTL to CTL (the model considers M as true)

5.5 Computational Complexity Analysis

5.5.1 Time Complexity of Model Checking 3v-TCTL through Transformation to TCTL

In this subsection, we will prove that the problem of model checking 3v-TCTL through the transformation to TCTL is P-complete in time. Generally, a problem is classified as P-complete if it is solvable in polynomial time, and any problem in P can be reduced to it in polynomial time. The assessment of model checking complexity is contingent upon the scale of both the model and the representation of the formula employed. Consequently, it is imperative to elucidate the chosen input representation and state how we measure its size.

Proposition 4. *(Boundedness of atomic propositions). Let $\mathcal{D}_T$ be the positive TCTL model obtained from $\mathbb{T}_M$ in Algorithm 6. Moreover, let $Atom(\mathcal{D}_T)$ be the number of atomic propositions in the model $\mathcal{D}_T$ and $Atom(\mathbb{T}_M)$ be the number of atomic propositions in model $\mathbb{T}_M$. We have: $Atom(\mathcal{D}_T) = 2 \times Atom(\mathbb{T}_M)$.*

Proof. The proof is straightforward from steps 7-9 in Algorithm 6. These steps introduce an additional atomic proposition in the model $\mathcal{D}_T$ for each atomic proposition. Thus, the number of atomic propositions in $\mathcal{D}_T$, $Atom(\mathcal{D}_T)$, is proportional to the number of atomic propositions in $\mathbb{T}_M$, $Atom(\mathbb{T}_M)$, with a scaling factor of 2. $\square$

Proposition 5. *(Boundedness of model transformation). Let $|\mathcal{D}_T|$ and $|\mathbb{T}_M|$ be the size of $\mathcal{D}_T$ and $\mathbb{T}_M$ respectively. $|\mathcal{D}_T|$ is linear with $|\mathbb{T}_M|$.*

Proof. As shown in Algorithm 6, the model $\mathcal{D}_T$ has the same number of states and relations as the model $\mathbb{T}_M$. The results flow then from Proposition 4. $\square$

Theorem 14. *(Explicit 3v-TCTL model checking: upper bound). The model checking problem of 3v-TCTL through transformation to TCTL can be solved in time $\mathcal{O}(|\mathbb{T}_M| \times |\phi|)$, where $|\mathbb{T}_M|$ and $|\phi|$ are the size of the model and length of the formula, respectively.*

Proof. 3v-TCTL extends TCTL. It is known from [24] that the model checking problem for TCTL is linear in the size of the model $\mathcal{D}_T$ and the length of the formula ψ. From Proposition 2, $|\mathcal{D}_T| = 2|\mathbb{T}_M|$ considering atomic variables as part of the state space. Moreover, from Procedure 1 explained with Algorithm 1, it is evident that the length of the transformed formula ψ equals the length of the original formula ϕ where the procedure only replaces each negated proposition $\neg x$ by $x-$, so the result. $\square$

Theorem 15. *(Explicit 3v-TCTL model checking: completeness). The model checking problem of 3v-TCTL through transformation to TCTL is P-complete.*

Proof. Membership in P (i.e., upper bound) follows from Theorem 14. Hardness in P (i.e., lower bound) follows by a reduction from model checking TCTL proved to be P-complete in [24]. $\square$

5.5.2 Space Complexity of Model Checking 3v-TCTL through Transformation to TCTL

Here, we prove that the complexity of 3v-TCTL model checking through the transformation to TCTL for concurrent programs [55] is PSPACE-complete. This finding indicates that there exists an algorithm capable of solving the problem within polynomial space relative to the size of the components comprising concurrent programs and the length of the formula undergoing model checking.

Theorem 16. *(Polynomial reduction of 3v-TCTL model checking: upper bound) Let $\sqsubseteq_{psr}$ denote the polynomial-space reduction. The problem of mv-model checking 3v-TCTL can be reduced to the problem of model checking TCTL in a polynomial space. Formally, mv-MC(3v-TCTL) $\sqsubseteq_{psr}$ MC(TCTL).*

Proof. Transforming the 3v-TCTL model and formula into the corresponding TCTL model and formula could be computed by a deterministic Turing Machine (TM) in space $O(log\ n)$ where n is the size of the input 3v-TCTL model, and polynomial space with regard to the length of the input formula. We used this notion to determine the upper bound of space complexity and indicate that the algorithm space increases logarithmically with the size of the input model.

TM performs one-by-one the following steps: (1) reads in the input tape a model of 3v-TCTL; (2) for each model, TM sequentially generates on the output tape the same states and the same valuations (mapping of atomic propositions); (3) replaces the value of any atomic proposition x equal to M ($x = M$) with T ($x = T$); (4) adds a fresh negation version for every atomic proposition in every state; (5) build the transitions between the obtained states where for the transitions $(s, s^{'})$ in the 3v-TCTL (input model), it writes the transitions in the set R_T. The same applies to the trust accessibility relations (AR_T) in the input model. The performed writing operations are clearly logarithmic in space as this transformation is conducted on-the-fly and step-by-step. Furthermore, as shown in the proof of Theorem 14, any TCTL formula is transformable into a non-negated TCTL formula with the same length. Based on this analysis, it is clear that all these transformations are polynomial spaces in the size of the inputs, so the result. □

Theorem 17. *3v-TCTL Model Checking for Concurrent Programs: Completeness). The space complexity of the 3v-TCTL model checking through transformation to TCTL for concurrent programs is PSPACE-complete with respect to both the size of the program components and the length of the formula.*

Proof. Given that model checking TCTL is proved PSPACE-complete for concurrent programs [24], it sets a lower bound for model checking 3v-TCTL at PSPACE. Notably, 3v-TCTL encompasses TCTL by incorporating both 3v-CTL modalities and trust modalities. The upper bound within PSPACE is given by Theorem 16, thereby affirming the result. □

5.5.3 Time Complexity of Model Checking 3v-TCTL through Transformation to CTL

In this subsection, we will prove that model checking 3v-TCTL through transformation to CTL is also P-complete in explicit models. We transform the 3v-TCTL logic into CTL in two steps. The first is from 3v-TCTL into TCTL, and this is addressed in the previous subsection. The second is from TCTL into CTL following the same strategy as in [24].

Proposition 6. *(Boundedness of model transformation). Let D_T be a positive CTL model obtained from the 3v-TCTL model $\mathbb{T}_M$ in Algorithm 7. Let $|D_T|$ and $|\mathbb{T}_M|$ be the size of D_T and $\mathbb{T}_M$, respectively. We have $|D_T| \leq 3|\mathbb{T}_M|$.*

Proof. Let $|A_M|$ be the number of trust accessibility relations in $\mathbb{T}_M$. We have $|\mathbb{T}_M| = |R_M| + |S_M| + |A_M|$ and $|D_T| = |S_T| + |R_T|$. As explained, each accessibility relation is translated into one additional state and two additional transitions in D_T. Based on this and using the same argument as in Proposition 4 (considering the atomic propositions part of the state space), we obtain $|D_T| \leq |R_M| + 2|S_M| + 3|A_M|$. In a typical scenario, multiple reachable states could exist between any given pair of states. Indeed, the other accessibility relations can be established by the already added states and transitions, which gives the result: $|D_T| \leq 3(|R_M| + |S_M| + |A_M|)$. □

Theorem 18. *(Explicit 3v-TCTL model checking: upper bound) The 3v-TCTL model checking problem can be solved in time $\mathcal{O}(|\mathbb{T}_M| \times |\phi|)$. Where $|\mathbb{T}_M|$ is the size of the model, and $|\phi|$ is the length of the formula.*

Proof. 3v-TCTL can be reduced to CTL. From [43], it is known that the model checking problem for CTL is linear in the size of the model and the length of the formula. From

Proposition 6, $|D_T| \leq 3|\mathbb{T}_M|$. The result follows from the fact that the length of the transformed CTL formula ϕ_1 from Algorithm 2 is linear with the length of the input trust formula ϕ as proved in Proposition 2 and Theorem 4 in [24]. □

Theorem 19. *(Explicit 3v-TCTL model checking: completeness). The model checking problem of 3v-TCTL through transformation to CTL is P-complete.*

Proof. Membership in P (i.e., upper bound) follows from Theorem 18. Hardness in P (i.e., lower bound) follows from the P-completeness of explicit model checking CTL [10]. □

5.5.4 Space Complexity of Model Checking 3v-TCTL through Transformation to CTL

We prove in this subsection that the complexity of 3v-TCTL model checking through the transformation to CTL for concurrent programs [55] is also PSPACE-complete.

Theorem 20. *(Polynomial reduction of 3v-TCTL model checking: upper bound) Let $\sqsubseteq_{psr}$ denote the polynomial-space reduction. The problem of mv-model checking 3v-TCTL can be reduced to the problem of model checking CTL in a polynomial space. Formally, mv-MC(3v-TCTL) $\sqsubseteq_{psr}$ MC(CTL).*

Proof. The transformation of the 3v-TCTL model and TCTL formula into the corresponding CTL model and formula could be computed by a deterministic Turing Machine (TM) in space $O(log\ n)$, where n is the size of the input 3v-TCTL model, and polynomial space concerning the length of the TCTL formula. Regarding the two transformed CTL models, for D_T, TM performs the following one-by-one steps: (1) reads in the input tape a model of 3v-TCTL; (2) generates in the output tape the same states with the same atomic propositions; (3) replaces the value of any atomic proposition x equal to M $(x = M)$ with T $(x = T)$; (4) adds a fresh negation version for every atomic proposition in every state; (5) build the transitions between the obtained states (R_T). For the trust accessibility relations $\sim_{i \to j}$, TM reads the relations between two given states in the input model, and for each one, it adds an intermediate state to the set of states (S_T). Each intermediate state is labeled with two fresh propositional variables: 1) α^{ij} presents the trust accessibility relation between agents i and j, and 2) χ indicates that the two transitions with the corresponding state do not already exist in the original model (3v-TCTL). All of the performed writing operations exhibit a clear logarithmic space complexity thanks to the on-the-fly, step-by-step nature of the transformation. Additionally, according to Proposition 2 in [24], it is demonstrated that any TCTL formula can be converted into a CTL formula with a length that is linearly bounded by that of the original input formula. These recursive transformations are evidently polynomial in space concerning the length of the input formula, thus establishing the theorem. □

Theorem 21. *3v-TCTL Model Checking for Concurrent Programs: Completeness). The space complexity of the 3v-TCTL model checking through transformation to CTL for concurrent programs is PSPACE-complete with respect to both the size of the program components and the length of the formula.*

Proof. Given that model checking CTL is PSPACE-complete for concurrent programs [43], it sets a lower bound for model checking 3v-TCTL at PSPACE-complete. Notably, 3v-TCTL encompasses CTL through TCTL by incorporating both 3v-CTL modalities and trust modalities. The upper bound within PSPACE is supported by Theorem 20, thereby affirming the result. □

Chapter 6

MV-Checker: A Software Tool for Multi-Valued Model Checking

This chapter presents the design and implementation of a new open-source and scalable software tool for modelling and verifying MAS, including IoT and IS systems with commitment and trust protocols under both uncertainty and inconsistency settings, using reduction-based multi-valued model checking techniques. The proposed tool is equipped with original and novel algorithms that transform our logics of multi-valued commitment (mv-CTLC) and multi-value trust (mv-TCTL) that we recently introduced to their classical two-valued commitment (CTLC) and trust (TCTL) logic versions as well as to Computational Tree Logic (CTL). Moreover, the tool transforms the mv-CTL to CTL, and it is applicable for the classical model checking by transforming the classical logics of trust and commitment to CTL.

6.1 Contributions

- Our main motivation is developing a Java-based tool named **MV-Checker**, which streamlines the process of verifying the properties of MAS, including IoT and IS systems, with trust and commitment within uncertain and inconsistent environments.

The tool is designed to perform the following functions: **(1)** Transforms our multi-valued logics of trust (3v-TCTL and 4v-TCTL) into TCTL and automatically interacts with the MCMSA^t model checker. **(2)** Transforms our multi-valued logics of commitment (3v-CTLC and 4v-CTLC) into CTLC and automatically interacts with the MCMSA+ model checker. **(3)** Additionally, the tool can transform these logics and mv-CTL logic into CTL and automatically interact with the NuSMV model checker. **(4)** Moreover, it allows directly verifying the classical CTL, CTLC and TCTL models. These functions make the **MV-Checker** an essential tool that efficiently and accurately verifies complex systems modeled in multi-valued logics of trust and commitment under uncertain and inconsistent environments.

- Technical contributions:
  - To check the effectiveness and efficiency of the proposed tool, we performed multiple verification experiments by modelling several IoT and intelligent applications involving trust and commitment protocols within uncertain and inconsistent environments.
  - We compared the developed approaches using the results obtained from transforming the mv-TCTL to CTL and classical TCTL. Additionally, we compared the results obtained from transforming the mv-CTLC to CTL and classical CTLC regarding scalability and accuracy.
  - We provide the packages of all our case studies containing 12 experiments with SMV, ISPL+, VISPL, mv-ISPL+ and mv-VISPL files. Additionally, we provide the source code, the Jar file of the tool, and the user manual document explaining the use of the proposed tool.

6.2 Comparative Analysis

We conduct comparative analyses with tools developed in prior related research studies to assess the proposed tool's performance and significant contributions to multi-valued model checking. The comparisons are based on several essential criteria, including (1) Underlying Modeling Framework, (2) Developed tool for implementing the proposed approach, (3) Notion of trust, (4) Notion of commitment, (5) Verification method, (6) Applicability to classical model checking, (7) Embedding of the multi-valued logic and (8) Applicability to multi-valued model checking. In Table 6.1, the first part presents an overview of tools developed in previous studies addressing the notion of trust and commitment. Some authors directly developed tools that handle trust and commitment logic modalities, while others devised tools that reduce their logics to CTL, CTL logic of time and knowledge (CTLK), and Action Restricted CTL (ARCTL). However, these tools are incapable of handling multi-valued logics. Additionally, the tools designed for trust-based models and formulae cannot be applied to commitment-based ones, and vice versa. In the second part of the table, we highlight proposed studies focusing on multi-valued model checking. Unfortunately, practical tools for applying this technique to system verification with multiple truth values are still limited. Moreover, these studies don't address the notion of trust and commitment in their logics. The last part of the table presents our tool, which is named MV-Checker. The checked criteria show that the tool fills the gap between the works presented in the first and the second parts. Our work applies to both the classical and multi-valued model checking techniques. Moreover, it handles verifying systems with CTL, mv-CTL, TCTL and mv-TCTL for trust, CTLC and mv-CTLC for commitment.

6.3 MV-Checker Tool: Internal Design

In this section, we produce a new verification tool named **MV-Checker** [1]. This tool is developed in Java for verifying multi-agent systems, including large and complex intelligent systems with trust and commitment protocols under uncertainty and inconsistency settings. The tool extends the Java toolkit developed by [24] for transforming the logic of trust (TCTL) to CTL for reusing the NuSMV model checker and automatically verifying multi-agent systems with trust protocols. More specifically, we produce a new software tool

[1]The tool and the case studies are available online at: `https://github.com/MV-checker/MV-Checker`

Table 6.1: Comparison between the features of the proposed tool and other tools with respect to the proposed criteria

Approach	Underlying Modeling Framework	**Imp.(Tool)**	Notion of Trust	Notion of Comm.	Verification Method	Applicable for MC	Multi-logic	Applicable for MV-MC
[29]	Vector-based interpreted systems	MCMAS^t	✓	—	Model Checking $TCTL$	✓	—	—
[22]	Vector-based interpreted systems	Reduction Java Toolkit (to CTL)	✓	—	Model Checking $TCTL^G$	✓	—	—
[24]	Vector-based interpreted systems	Reduction Java Toolkit (to CTL)	✓	—	Model checking BT	✓	—	—
[36]	Interpreted system	Reduction Com-2-CWB tool (to CTLK and ARCTL)	—	✓	Model Checking CTLC	✓	—	—
[38]	Interpreted system	Reduction Com-2-CWB to ARCTL	—	✓	Model checking $CTLC^+$	✓	—	—
[31]	Interpreted system	MCMAS+	—	✓	Model checking $CTLC^{cc}$	✓	—	—
[17]	Multi-valued Kripke structure	—	—	—	Multi-valued Model Checking for mv-CTL	✓	✓	✓
[53]	Multi-valued finite state model	—	—	—	Model checking for $mv - CTL^*$	✓	✓	✓
[58]	Multi-valued Kripke structure	—	—	—	Model checking mv-CTL	✓	✓	✓
[50]	Multi-valued concurrent game structure (CGS)	STV	—	—	Model Checking $ATL^*_{\rightarrow}$	✓	✓	✓
Our work	classical and Multi-valued interpreted system	mv-Checker (atomically interacts with MCMAS and NuSMV)	✓	✓	Multi-valued Model Checking for CTL,mv-CTL, TCTL, mv-TCTL, CTLC and mv-CTLC	✓	✓	✓

that verifies multi-valued trust and commitment systems models designed under uncertain or inconsistent sources of knowledge. The models are formally written in new languages named mv-ISPL+ for commitment and mv-VISPL for trust. In these models, the atomic propositions in states take truth values over a given lattice depending on the purpose of verifying the system under investigation. The tool transforms the multi-valued models to their classical counterparts and automatically calls MCMAS+ for commitments and MCMAS^t for trust. Additionally, it transforms these models to CTL and automatically calls NuSMV checker. Furthermore, the tool transforms the mv-CTL to CTL, and also allows direct and semi-direct (from classical commitment and trust to CTL) verification of classical trust and commitment models.

6.3.1 Use Case Diagrams

The use case diagram of MV-Checker is divided into two main use cases. Figure 6.1 shows the tool's functions interacting with MCMAS+ for commitment and MCMAS^t for trust. In this figure, the user starts by choosing to upload one of the following files:

1. Upload ISPL+ for classical CTLC of commitment, then launch MCMAS+ to verify the uploaded model (direct verification).

2. Upload VISPL for classical TCTL of trust, then launch MCMAS^T to verify the uploaded model (direct verification).

3. Upload 3v-ISPL+ for 3v-CTLC of commitment to reason about uncertainty. Then, the tool transforms this model into two ISPL+ for classical CTLC models. One replaces the value M of every atomic proposition with the value T, and the other replaces it with the value F. Finally, launch MCMAS+ over each model to have the verification results.

4. Upload 4v-ISPL+ for 4v-CTLC of commitment to reason about inconsistency. Then, the tool transforms this model into two ISPL+ for classical CTLC models. One replaces the values FT and FF of every atomic proposition with the value F and replaces the values TT and TF with T. The other model replaces the value TF and FF of every atomic proposition with the value F and replaces the values TT and FT with T. Finally, launch MCMAS+ over each model to have the verification results.

5. Upload 3v-VISPL for 3v-TCTL of trust to reason about uncertainty. Then the tool applies the same strategy in 2 by transforming the uploaded model into two classical VISPL models and launches MCMAS^t over each model.

6. Upload 4v-VISPL for 4v-TCTL of trust to reason about inconsistency. Then the tool applies the same strategy in 3 by transforming the uploaded model into two classical VISPL models and launches MCMAS^T over each model.

Figure 6.2 shows the second main use case. In this figure, the tool transforms the uploaded files into CTL and automatically calls NuSMV checker. The functions of this part are as follows:

1. Upload 3v-ISPL+ for 3v-CTLC of commitment to reason about uncertainty. Then, the tool transforms this model into two ISPL+ for classical CTLC models. One replaces the value M of every atomic proposition with the value T, and the other replaces it with the value F. Next, the tool transforms each model into SMV model (CTL model). Finally, launches NuSMV over each CTL model to obtain the verification results.

2. Upload 4v-ISPL+ for 4v-CTLC of commitment to reason about inconsistency. Then, the tool transforms this model into two ISPL+ for classical CTLC models. One replaces the values FT and FF of every atomic proposition with the value F and replaces the values TT and TF with T. The other model replaces the value TF and FF of every atomic proposition with the value F and replaces the values TT and FT with T. Next, the tool transforms each model into SMV model (CTL model). Finally, launches NuSMV over each CTL model to have the verification results.

3. Upload 3v-VISPL for 3v-TCTL of trust to reason about uncertainty. Then, the tool applies the same strategy in 1 by transforming the uploaded model into two classical VISPL models. Then, it transforms each model into CTL model and launches NuSMV over each.

4. Upload 4v-VISPL for 4v-TCTL of trust to reason about inconsistency. Then, the tool applies the same strategy in 2 by transforming the uploaded model into two classical VISPL models. Then, it transforms each model into CTL model and launches NuSMV over each.

5. The tool also allows the transformation of the 3v-CTL and 4v-CTL to the classical versions. This transformation is used when reasoning about uncertainty or inconsistency over systems where trust and commitment protocols are not considered in the system's behaviors.

6. Additionally, the tool is capable of performing the direct verification of CTL models.

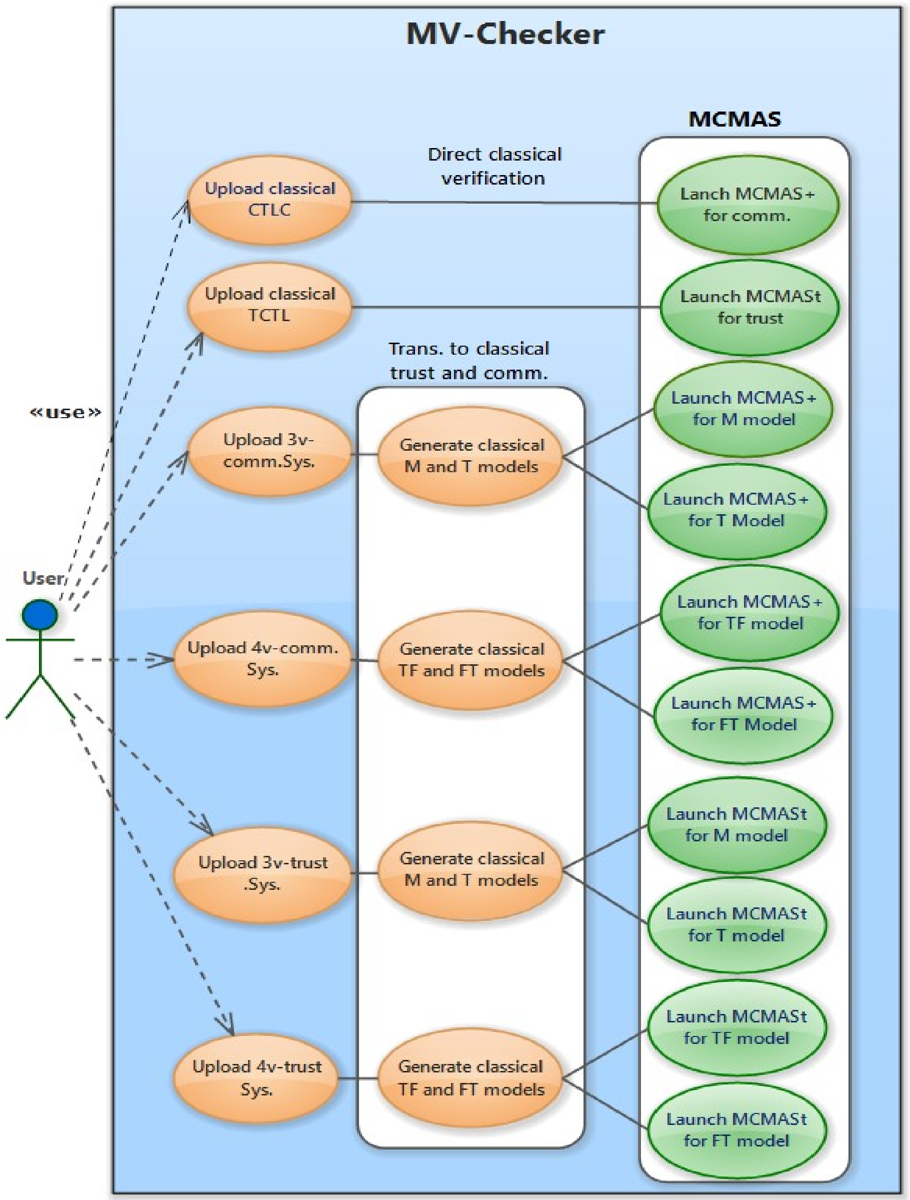

MV-Checker
MCMAS
User
«use»
Upload classical CTLC
Upload classical TCTL
Upload 3v-comm.Sys.
Upload 4v-comm. Sys.
Upload 3v-trust .Sys.
Upload 4v-trust Sys.
Direct classical verification
Trans. to classical trust and comm.
Generate classical M and T models
Generate classical TF and FT models
Generate classical M and T models
Generate classical TF and FT models
Lanch MCMAS+ for comm.
Launch MCMASt for trust
Launch MCMAS+ for M model
Launch MCMAS+ for T Model
Launch MCMAS+ for TF model
Launch MCMAS+ for FT Model
Launch MCMASt for M model
Launch MCMASt for T model
Launch MCMASt for TF model
Launch MCMASt for FT model

Figure 6.1: A use case diagram of the MV-Checker software tool interacts with MCMAS+ for commitment and MCMAS^T for trust

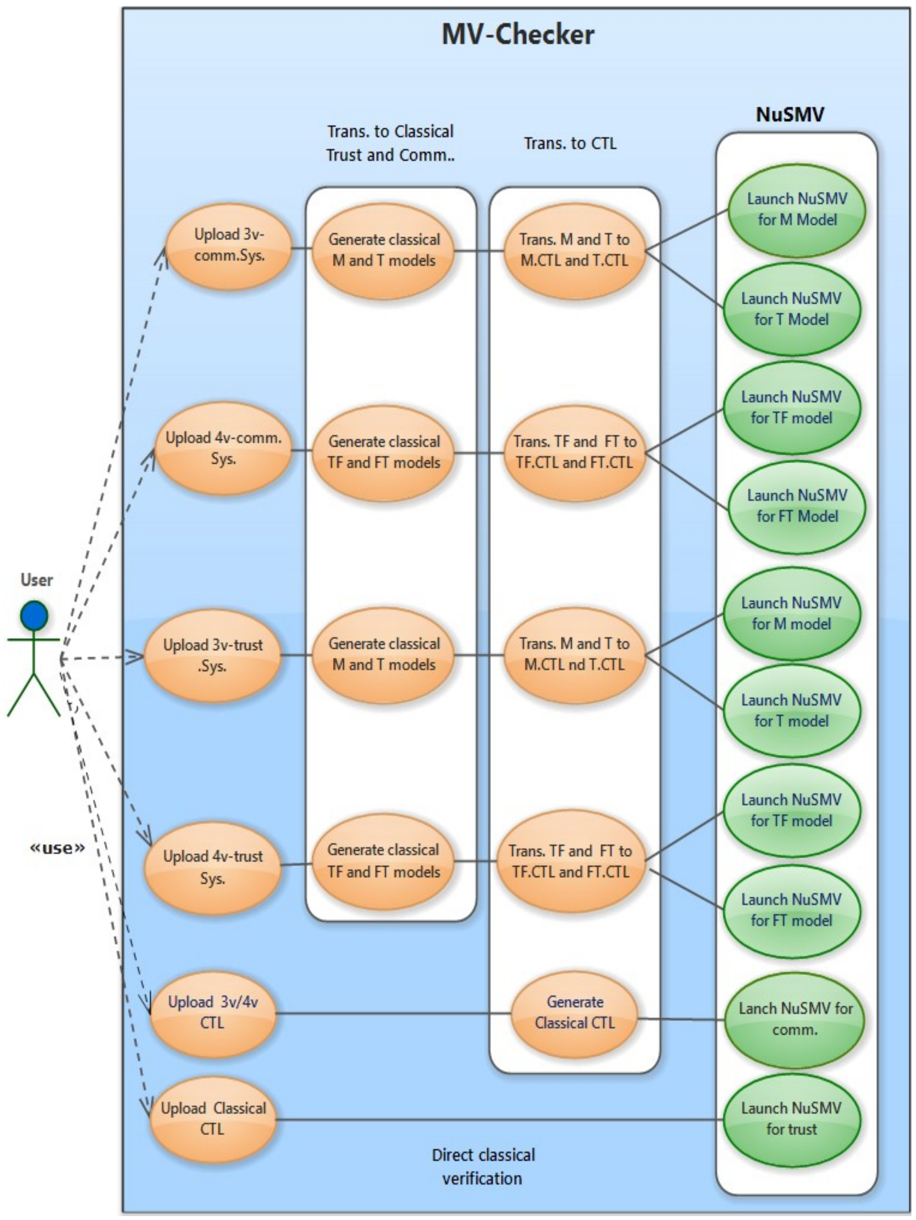

MV-Checker
NuSMV
Trans. to Classical Trust and Comm..
Trans. to CTL
User
«use»
Upload 3v-comm.Sys.
Generate classical M and T models
Trans. M and T to M.CTL and T.CTL
Launch NuSMV for M Model
Launch NuSMV for T Model
Upload 4v-comm. Sys.
Generate classical TF and FT models
Trans. TF and FT to TF.CTL and FT.CTL
Launch NuSMV for TF model
Launch NuSMV for FT Model
Upload 3v-trust .Sys.
Generate classical M and T models
Trans. M and T to M.CTL nd T.CTL
Launch NuSMV for M model
Launch NuSMV for T model
Upload 4v-trust Sys.
Generate classical TF and FT models
Trans. TF and FT to TF.CTL and FT.CTL
Launch NuSMV for TF model
Launch NuSMV for FT model
Upload 3v/4v CTL
Generate Classical CTL
Lanch NuSMV for comm.
Upload Classical CTL
Launch NuSMV for trust
Direct classical verification

Figure 6.2: A use case diagram of the MV-Checker software tool interacts with NuSMV

6.3.2 Sequence Diagrams

To have a deeper insight into the proposed system functions and determine the classes of its code, we need to have a close picture of the data exchange between the user, MV-Checker, MCMAS+, MCMAS^t and NuSMV. The sequence diagram allows an internal view of the data flow during the system usage. We divide the sequence diagram of the system into three main parts. The first (Figure 6.4) shows the data exchange among the user, MV-Checker, and MCMAS+ and MCMAS^t while transforming the mv-CTLC and mv-TCTL models to their classical counterparts to be verified using the two checkers. The second (Figure 6.5) shows the data exchange among the user, MV-Checker, and NuSMV while transforming these logics to CTL to be verified using NuSMV. The third part (Figure 6.3) shows the additional functionalities of verifying the classical CTLC and TCTL using the related versions of MCMAS, in addition to verifying the classical CTL directly or after transforming it from 3v/4v-CTL.

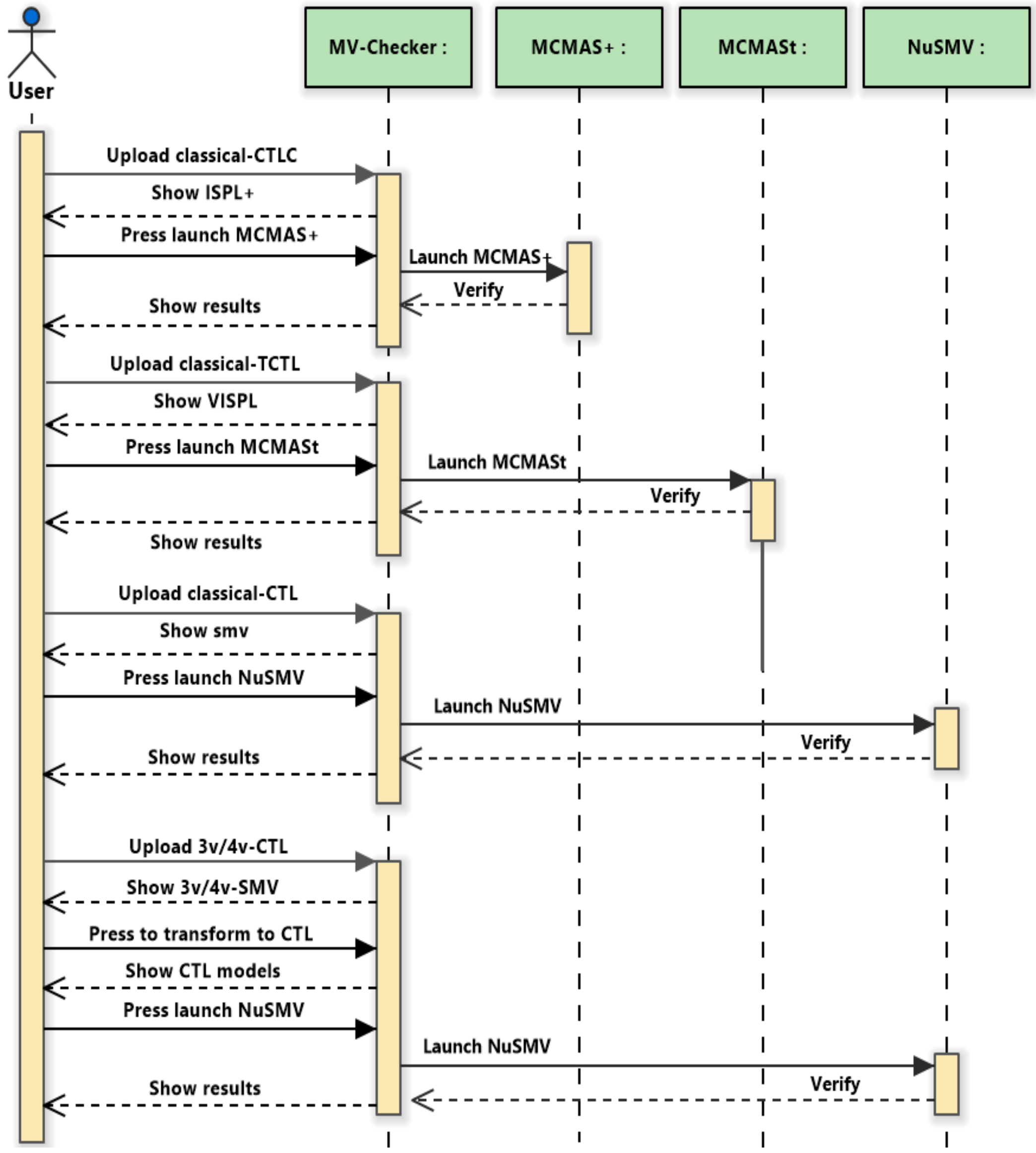


Figure 6.3: A sequence diagram of the MV-Checker software tool (the additional functionalities of verifying the classical CTLC, TCTL and CTL directly or after transforming it from 3v/4v-CTL)

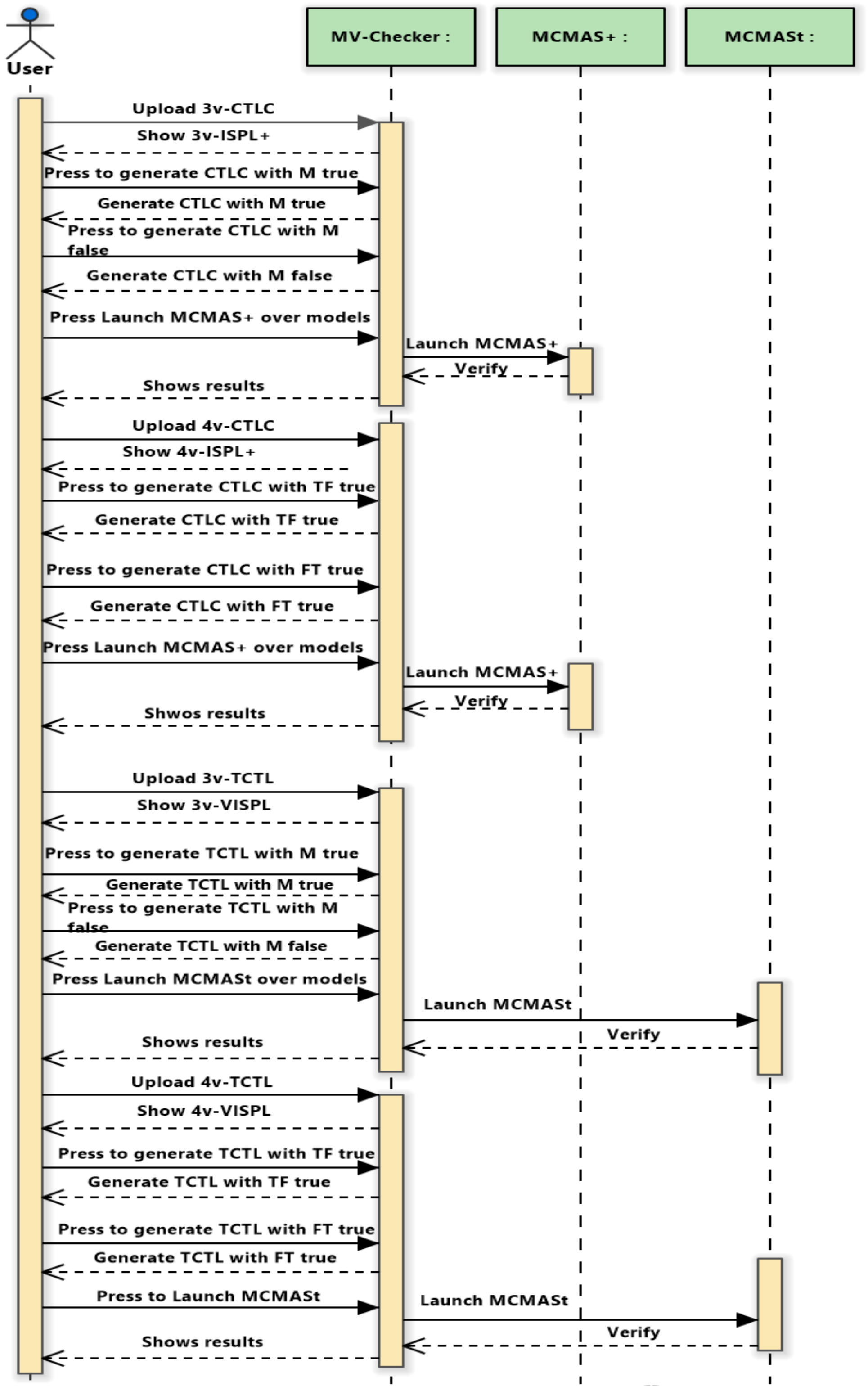

User
MV-Checker :
MCMAS+ :
MCMASt :
Upload 3v-CTLC
Show 3v-ISPL+
Press to generate CTLC with M true
Generate CTLC with M true
Press to generate CTLC with M false
Generate CTLC with M false
Press Launch MCMAS+ over models
Launch MCMAS+
Verify
Shows results
Upload 4v-CTLC
Show 4v-ISPL+
Press to generate CTLC with TF true
Generate CTLC with TF true
Press to generate CTLC with FT true
Generate CTLC with FT true
Press Launch MCMAS+ over models
Launch MCMAS+
Verify
Shwos results
Upload 3v-TCTL
Show 3v-VISPL
Press to generate TCTL with M true
Generate TCTL with M true
Press to generate TCTL with M false
Generate TCTL with M false
Press Launch MCMASt over models
Launch MCMASt
Verify
Shows results
Upload 4v-TCTL
Show 4v-VISPL
Press to generate TCTL with TF true
Generate TCTL with TF true
Press to generate TCTL with FT true
Generate TCTL with FT true
Press to Launch MCMASt
Launch MCMASt
Verify
Shows results

Figure 6.4: A sequence diagram of the MV-Checker software tool (data exchange between the user, MV-Checker and MCMASs)

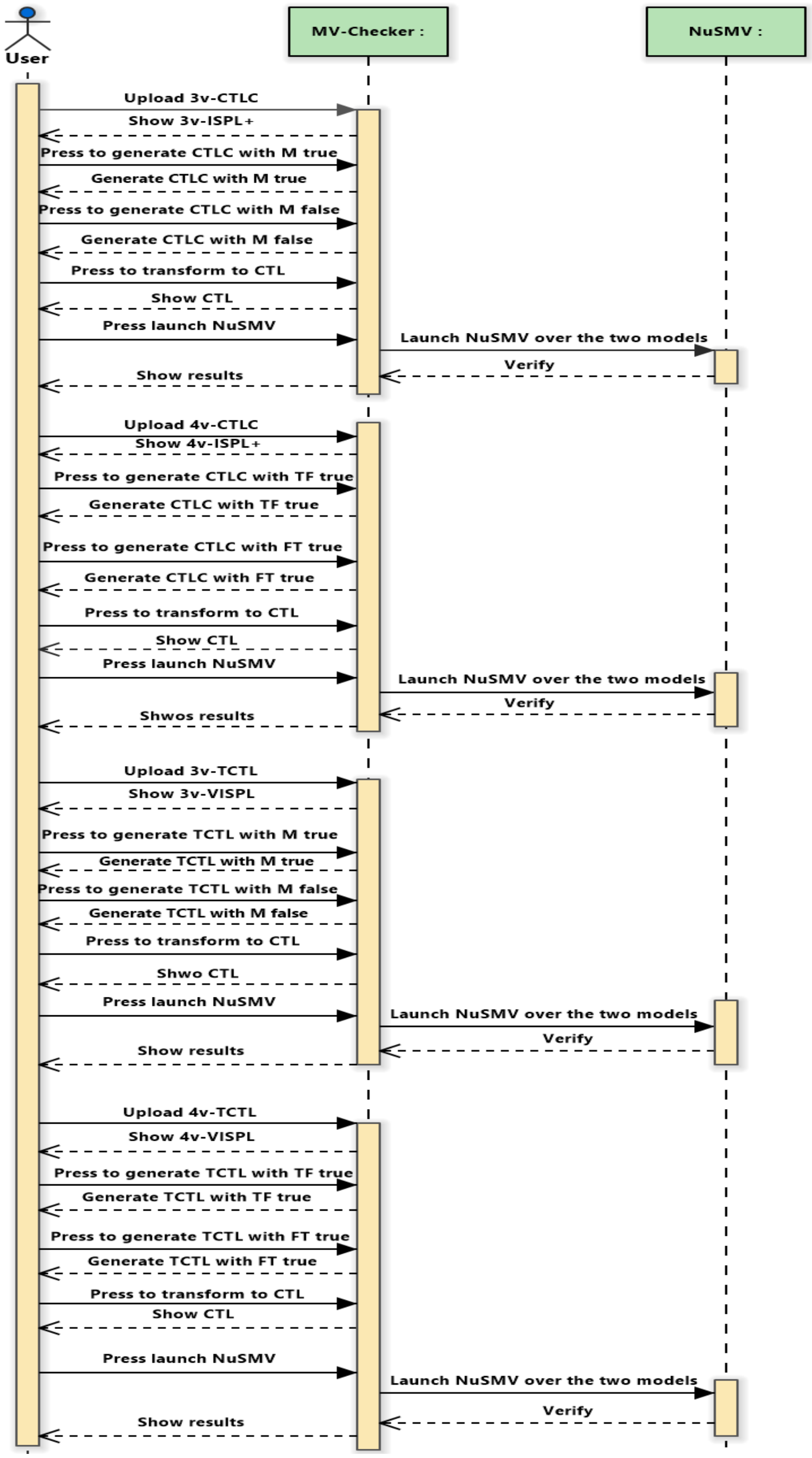

User
MV-Checker :
NuSMV :
Upload 3v-CTLC
Show 3v-ISPL+
Press to generate CTLC with M true
Generate CTLC with M true
Press to generate CTLC with M false
Generate CTLC with M false
Press to transform to CTL
Show CTL
Press launch NuSMV
Launch NuSMV over the two models
Verify
Show results
Upload 4v-CTLC
Show 4v-ISPL+
Press to generate CTLC with TF true
Generate CTLC with TF true
Press to generate CTLC with FT true
Generate CTLC with FT true
Press to transform to CTL
Show CTL
Press launch NuSMV
Launch NuSMV over the two models
Verify
Shwos results
Upload 3v-TCTL
Show 3v-VISPL
Press to generate TCTL with M true
Generate TCTL with M true
Press to generate TCTL with M false
Generate TCTL with M false
Press to transform to CTL
Shwo CTL
Press launch NuSMV
Launch NuSMV over the two models
Verify
Show results
Upload 4v-TCTL
Show 4v-VISPL
Press to generate TCTL with TF true
Generate TCTL with TF true
Press to generate TCTL with FT true
Generate TCTL with FT true
Press to transform to CTL
Show CTL
Press launch NuSMV
Launch NuSMV over the two models
Verify
Show results

Figure 6.5: A sequence diagram of the MV-Checker software tool (data exchange between the user, MV-Checker and NuSMV)

6.3.3 Class Diagram

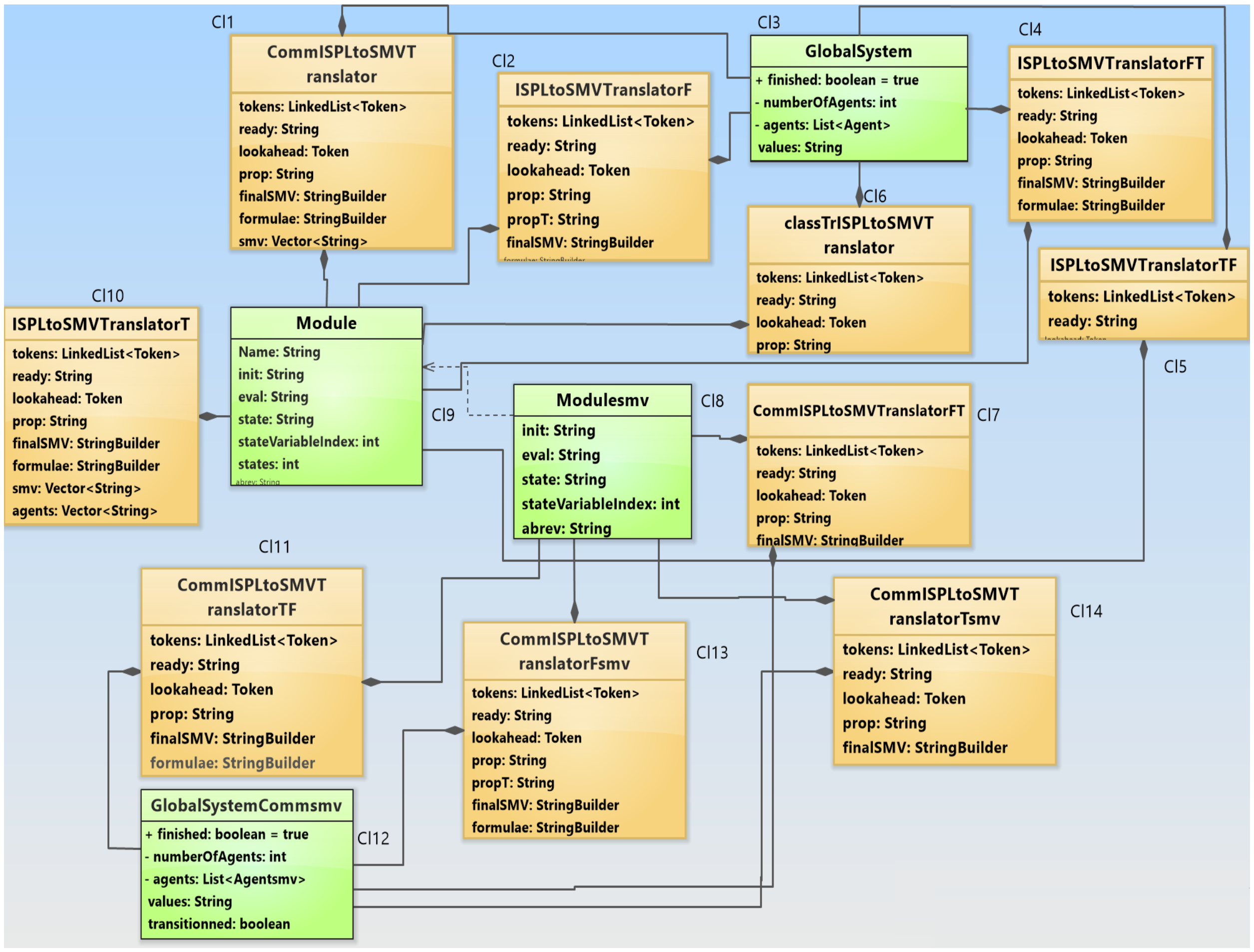


Figure 6.6: A class diagram of the MV-Checker software tool(Main classes)

The MV-Checker tool, implemented in Java, is composed of 48 distinct classes that comprise its code. In this section, we include and explain 14 main classes linked by composition relations. In Figure 6.6, class Cl1 calls class Cl3 to establish the transformation of the classical model of CTLC for commitment to SMV. The same class calls Cl9 to establish the names of states and agent variables. The class also adds the new accessibility relations after translating to CTL. After transforming the 3v-TCTL of trust to two classical TCTL models, class Cl2 and Cl10 call Cl3 and Cl9 to transform the TCTL model that considers M as F and the one that considers M as T, respectively, to CTL. Class Cl4 performs similar functions after transforming the 4v-TCTL model to classical TCTL, which considers only

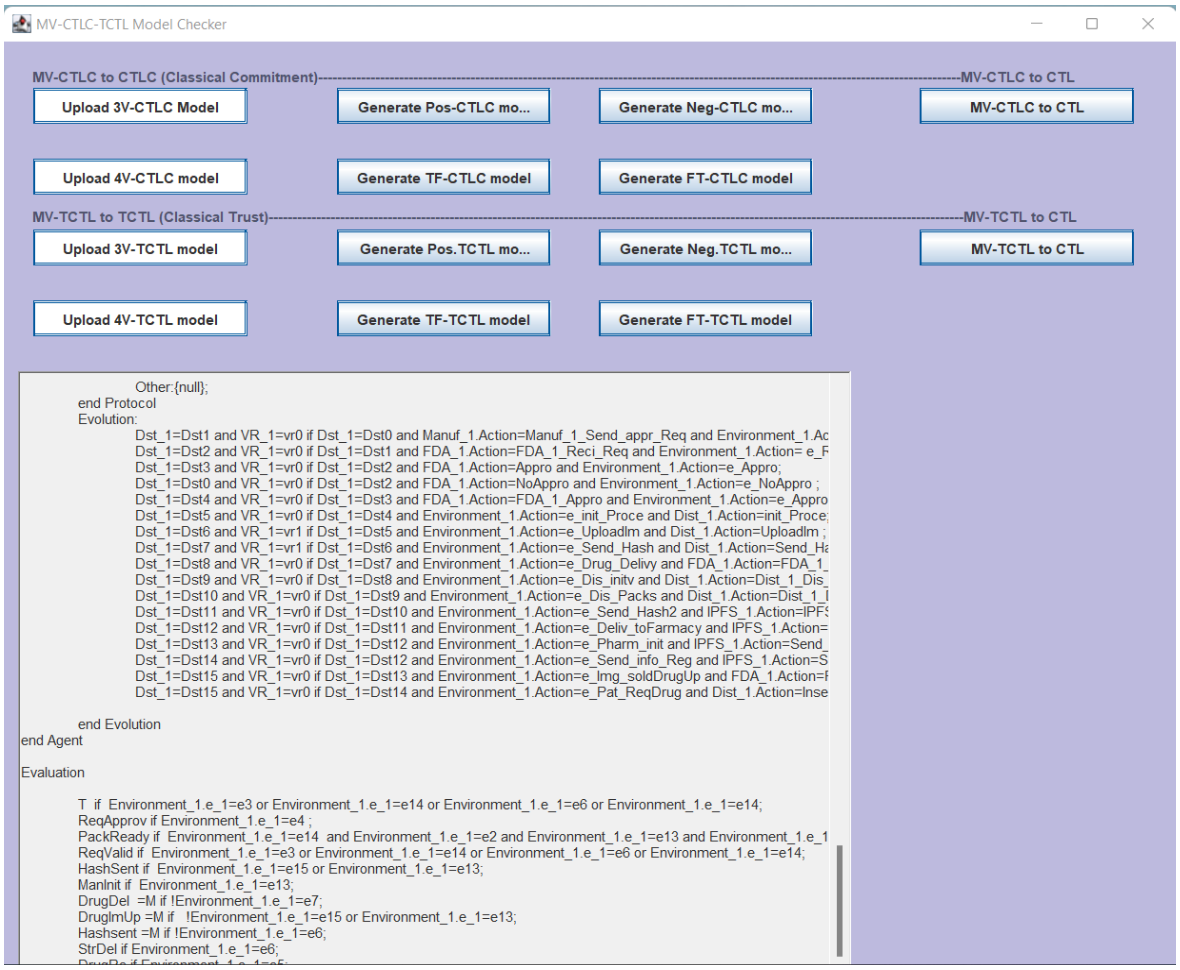
FT and TT as T. Then, it transforms this model to CTL. As well, Cl5 calls Cl3 and Cl9 to transform the classical TCTL (obtained from the 4v-TCTL), which considers only TF and TT as T, to CTL. Cl6 directly transforms the classical TCTL to CTL through Cl3 and Cl9. Classes Cl7 and Cl11 call Cl8 and Cl12 to transform the two classical CTLC commitment models generated from the 4v-CTLC to CTL. Classes Cl13 and Cl14 call Cl8 and Cl12 to transform the two classical CTLC commitment models generated from the 3v-CTLC to CTL. Cl13 and Cl14 call the same last two classes (Cl8 and Cl12) to transform the classical two CTLC models obtained from the 3v-CTLC, to CTL.

6.4 MV-Checker Interface (Main Screen)

Figure 6.7 shows the main screen of the **MV-Checker**[2]. As shown in the figure, the screen contains seven buttons on the upper part for transforming the mv-CTLC of commitment to CTLC and CTL, and the other seven buttons are for transforming the mv-TCTL of trust to TCTL and CTL, in addition to a text area. These buttons perform the following tasks: The "Upload 3v-CTLC model" button allows uploading the corresponding 3v-ISPL+ file, where the content of this file will appear in the text area. If the uploaded file is incorrect, a separate screen will prompt the user to upload the appropriate file. The other "Upload" buttons (distinguished by their Gray color) perform similar functions to uploading the desired versions of the ISPL files. The two buttons, "Generate Pos-CTLC model" and "Generate Neg-CTLC model", generate two separate screens to transform the 3v-CTLC model to two classical CTLCs to be verified by MCMAS+. Similarly, the two buttons, "Generate TF-CTLC" and "Generate FT-CTLC", facilitate the transformation of the 4v-CTLC to its classical versions. The "mv-CTLC to CTL" button generates another main screen containing all the functions to transform the 3v/4v-CTLC models to CTL models to be verified by NuSMV. The remaining buttons for trust function in a similar way to transfer the models with mv-logic of trust to their classical versions and to CTL to be verified by MCMAS^t and NuSMV, respectively. More explanation of the tool interfaces will

[2]The tool and the case studies are available online at: `https://github.com/MV-checker/MV-Checker`

be provided in the next section.



```
                Other:{null};
        end Protocol
        Evolution:
                Dst_1=Dst1 and VR_1=vr0 if Dst_1=Dst0 and Manuf_1.Action=Manuf_1_Send_appr_Req and Environment_1.Ac
                Dst_1=Dst2 and VR_1=vr0 if Dst_1=Dst1 and FDA_1.Action=FDA_1_Reci_Req and Environment_1.Action= e_F
                Dst_1=Dst3 and VR_1=vr0 if Dst_1=Dst2 and FDA_1.Action=Appro and Environment_1.Action=e_Appro;
                Dst_1=Dst0 and VR_1=vr0 if Dst_1=Dst2 and FDA_1.Action=NoAppro and Environment_1.Action=e_NoAppro ;
                Dst_1=Dst4 and VR_1=vr0 if Dst_1=Dst3 and FDA_1.Action=FDA_1_Appro and Environment_1.Action=e_Appro
                Dst_1=Dst5 and VR_1=vr0 if Dst_1=Dst4 and Environment_1.Action=e_init_Proce and Dist_1.Action=init_Proce;
                Dst_1=Dst6 and VR_1=vr1 if Dst_1=Dst5 and Environment_1.Action=e_UploadIm and Dist_1.Action=UploadIm ;
                Dst_1=Dst7 and VR_1=vr1 if Dst_1=Dst6 and Environment_1.Action=e_Send_Hash and Dist_1.Action=Send_Ha
                Dst_1=Dst8 and VR_1=vr0 if Dst_1=Dst7 and Environment_1.Action=e_Drug_Delivy and FDA_1.Action=FDA_1_
                Dst_1=Dst9 and VR_1=vr0 if Dst_1=Dst8 and Environment_1.Action=e_Dis_initv and Dist_1.Action=Dist_1_Dis_
                Dst_1=Dst10 and VR_1=vr0 if Dst_1=Dst9 and Environment_1.Action=e_Dis_Packs and Dist_1.Action=Dist_1_I
                Dst_1=Dst11 and VR_1=vr0 if Dst_1=Dst10 and Environment_1.Action=e_Send_Hash2 and IPFS_1.Action=IPFS
                Dst_1=Dst12 and VR_1=vr0 if Dst_1=Dst11 and Environment_1.Action=e_Deliv_toFarmacy and IPFS_1.Action=
                Dst_1=Dst13 and VR_1=vr0 if Dst_1=Dst12 and Environment_1.Action=e_Pharm_init and IPFS_1.Action=Send_
                Dst_1=Dst14 and VR_1=vr0 if Dst_1=Dst12 and Environment_1.Action=e_Send_info_Reg and IPFS_1.Action=S
                Dst_1=Dst15 and VR_1=vr0 if Dst_1=Dst13 and Environment_1.Action=e_Img_soldDrugUp and FDA_1.Action=F
                Dst_1=Dst15 and VR_1=vr0 if Dst_1=Dst14 and Environment_1.Action=e_Pat_ReqDrug and Dist_1.Action=Inse

        end Evolution
end Agent

Evaluation

        T  if  Environment_1.e_1=e3 or Environment_1.e_1=e14 or Environment_1.e_1=e6 or Environment_1.e_1=e14;
        ReqApprov if Environment_1.e_1=e4 ;
        PackReady if  Environment_1.e_1=e14  and Environment_1.e_1=e2 and Environment_1.e_1=e13 and Environment_1.e_1
        ReqValid if  Environment_1.e_1=e3 or Environment_1.e_1=e14 or Environment_1.e_1=e6 or Environment_1.e_1=e14;
        HashSent if  Environment_1.e_1=e15 or Environment_1.e_1=e13;
        ManInit if  Environment_1.e_1=e13;
        DrugDel  =M if !Environment_1.e_1=e7;
        DrugImUp =M if   !Environment_1.e_1=e15 or Environment_1.e_1=e13;
        Hashsent =M if !Environment_1.e_1=e6;
        StrDel if Environment_1.e_1=e6;
```

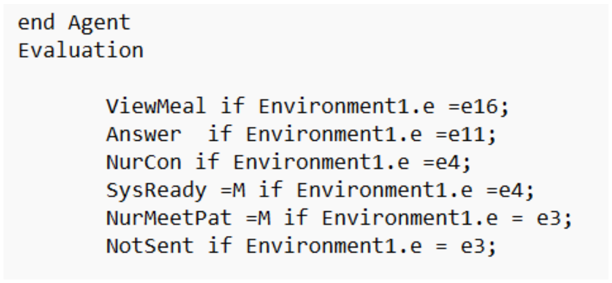
Figure 6.7: The main screen shows the uploading of the 3v-TCTL model encoded in 3v-VISPL as an input language

Chapter 7

Application Domains and Case Studies

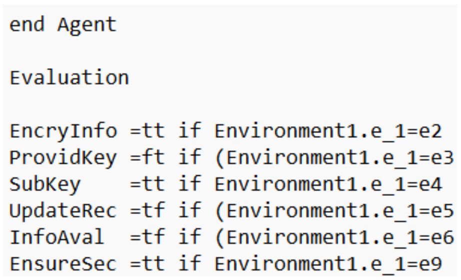
In this chapter, we present our experiments applied to multiple case studies covering IoT and IS systems with trust and commitment protocols. We model the systems under investigation using our logics to capture the presence of uncertainty and inconsistency in these models. Then, we assign for each model a set of specifications to be checked and verify whether the system satisfies its specifications or not. Next, we translate the models and their specifications into formal multi-valued ISPL language files mv-ISPL+ for commitment and mv-VISPL for trust, as shown in Figures 7.1, (a) and (b), respectively. In (a), the atomic propositions take truth values over lattice L_3, which means we miss information about the ones with M. In (b), the atomic propositions take truth values over lattice L_4, which means we have inconsistency about the ones with TF and FT and agreement about the ones with values TT and FF.

Finally, we call these files through the new tool MV-Checker and perform the verification processes. Ultimately, we report our results and make multiple comparison analyses regarding the execution times and memory in use. The comparison analysis allows us to evaluate the efficiency of our approaches and the performance of our tool, where we compare its performance with another tool in the same field. Based on the final results, we set our



```
end Agent
Evaluation

       ViewMeal if Environment1.e =e16;
       Answer  if Environment1.e =e11;
       NurCon if Environment1.e =e4;
       SysReady =M if Environment1.e =e4;
       NurMeetPat =M if Environment1.e = e3;
       NotSent if Environment1.e = e3;
```

(a) 3v-ISPL+



```
end Agent

Evaluation

EncryInfo =tt if Environment1.e_1=e2
ProvidKey =ft if (Environment1.e_1=e3
SubKey    =tt if Environment1.e_1=e4
UpdateRec =tf if (Environment1.e_1=e5
InfoAval  =tf if (Environment1.e_1=e6
EnsureSec =tt if Environment1.e_1=e9
```

(b) 4v-VISPL

Figure 7.1: 3v-ISPL+ and 4v-VISPL input languages

future work directions to improve the recent work and come out with future studies. [1].

7.1 Modeling and Verifying 3v-CTL and Commitment Systems with Uncertainty

7.1.1 Case study 1: A 3v-CTL Smart Home System with Uncertainty

We model a Smart Home system with multi-source data as shown in Figure 7.2. This model shows a smart home system's behaviours where the user has control via smartphone over smart devices: a washing machine, thermostat, door-lock, doorbell, motion detector, front-door lighting, smog alarm and fire alarm connected with a Fire department. The system generally functions as follows: The user has control over the mentioned smart devices through an application installed in a smartphone. When the smart fire alarm detects a fire, it sends a signal to the user and the fire department. At the same time, in case of an emergency, the application sends an alert to the fire department. Moreover, the smart washing machine sends a reminder at a specific time scheduled by the user for loading the machine with clothes. The system is also connected to a security camera and motion sensor installed on the front door. When the sensor detects a motion, the security camera imminently opens and sends a picture to the user. In addition, the front light should turn on according to the darkness level in the home area. Then, the user decides whether to open

[1]The tool and the case studies are available online at:
`https://github.com/MV-checker/MV-Checker`

the door or not. Furthermore, the system is connected to a smart thermostat that opens or closes the air conditioner or the heater based on the room temperature. From Figure 7.2, it is notable that the system has uncertainty on the atomic propositions (*frontlight*, *load*, *voiceRec*, *Above*, *Below* and *DoorBell*) which means we miss information about these atomic propositions in their states. At the same time, we have certain information about the ones with absolute T and F values. We excluded the variables with the value F to simplify the system.

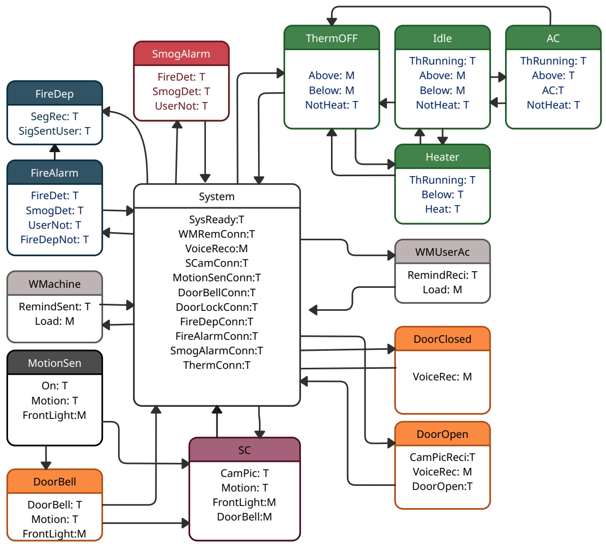


Figure 7.2: A Smart Home system scenario

System Specifications

We check seven properties in our system: (1) safety, meaning that no bad situations will happen while executing the system (φ_1 and φ_2 in the following list). We expressed these properties in two different ways to check the accuracy of our results and the efficiency of our algorithm. In φ_1, we check a weak safety, so we use the quantifiers AF for checking a condition that needs to hold in the future (in all future paths) and not necessarily for the entire global behavior of the system. In other words, this safety property is concerned with what happens eventually (in the future) and not throughout all possible system paths. In φ_2, we check the strong safety, so we use the quantifiers AG to check for a condition that should hold true for every possible path in the system, from the initial state to all future states. The property aims to ensure that in every execution of the system, there is no scenario where a fire signal is received, but the fire department fails to connect to the system. (2) Reachability expresses that the system will eventually reach a particular state during its execution (φ_3, φ_4 and φ_5). The formula φ_3 expresses the usual (unconditional) reachability with the operator EF. It checks that there exists at least one path in the system's behavior where the system will reach a particular situation, and this must hold true for all possible paths globally. The formulas φ_4 and φ_5 express a conditional reachability with the operator AX. They check that a situation is reachable in all next states if a condition holds. (3) Liveness, expresses that something good will eventually happen (φ_6 and φ_7. The property φ_6 asserts that there is a possible execution in the future where the desired conditions are met. The property φ_7 describes a desirable behavior that should happen in all possible system behaviors. It ensures that the camera's response to the doorbell is prompt and timely, making it a liveness property. Below is the list of properties where the meanings of the atomic propositions in the logic formula are indicated in bold in the corresponding formulae explanations.

- $\varphi_1 = AF\neg(NotCamrec \wedge DoorOpen)$ means it is not the case that **the user doesn't receive the security camera picture** when **the door opens**.
- $\varphi_2 = AG\neg(SigRec \wedge EX(NotFireDpartConn))$ means it is not the case that **a fire signal is received** but **the fire department doesn't connect to the system**.
- $\varphi_3 = AG(EF(SysReady))$ means in all paths globally, **the system will be ready**.
- $\varphi_4 = AG(Motion \implies AX(FrontLight))$ means whenever **a motion is detected**, the **front light immediately opens**.
- $\varphi_5 = AG(SmogDet \implies AX(FireDet))$ means whenever the smog alarm **detects a smog**, it immediately **alerts the fire department**.

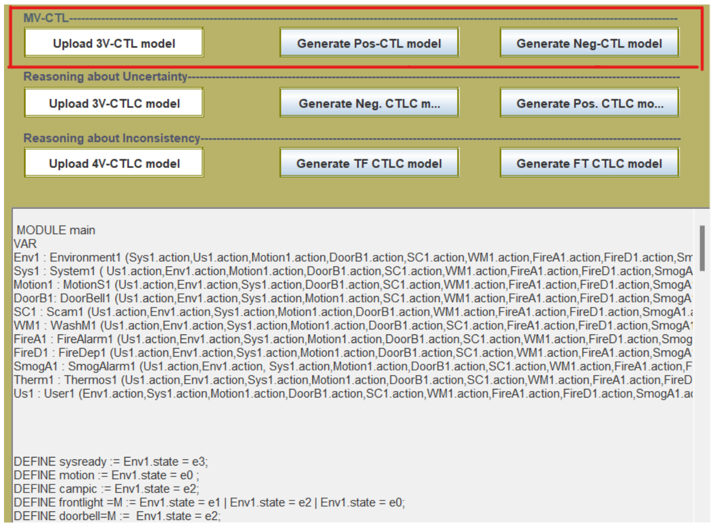
- $\varphi_6 = EF(above \wedge (ThRunning \wedge NotHeat))$ means there is a possible execution where the **temperature is above** the desired threshold while **the thermostat is ON** and **the heater is Off**.
- $\varphi_7 = AG(DoorBell \implies EX(CamPic))$ means whenever **the doorbell rings**, the security camera immediately **takes a picture**.

System Verification

We use our new tool **MV-Checker** to implement the reduction approach. This tool allowed for the first full implementation of the reduction algorithm (Algorithm 1). Technically, from a 3-valued Smart Home model uploaded in Figure 7.3, the tool derives 2-valued models as shown in Figure 7.4: a classical CTL model called positive cut, considers M as *True*; and a classical CTL model called negative cut, considers M as *False*. Then, the tool calls NusMV model checker over each model and checks if the *positive cut* yields F then, the result returns "*False*"; if not, it checks the *negative cut*, if yields T, the result is "*True*"; otherwise, the result will be "*Maybe*". For example, the verification result of property $\varphi_4 = AG(Motion \implies AX(FrontLight))$ is obtained as follows: calling NuSMV model checker over the negative cut gave (False) for this formula, then we checked the positive cut that gave the result (True) as shown in Figure 7.5. From these results, we conclude that it is uncertain whether this formula is satisfied in the system. This is evident because we lack information about the automatic proposition "*FrontLight*" in the original 3v-CTL model.

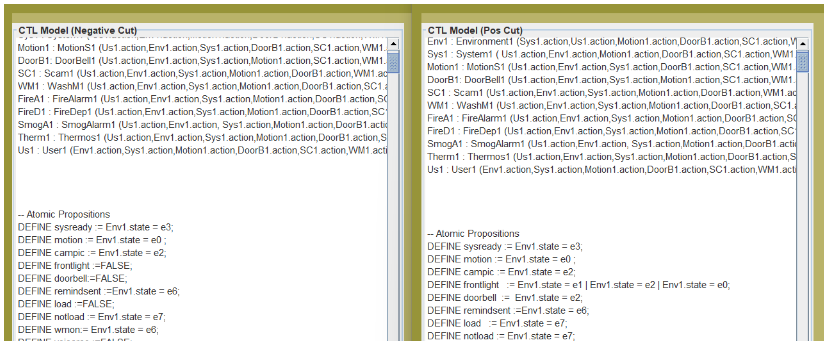
Verification Results

Table 7.1 illustrates the verification results using the reduction approach implemented by the new tool. The table shows that properties $(\varphi_1, \varphi_2, \varphi_3)$ are satisfied in the model because the verification of two two-valued models gave us (T). In contrast, the properties $(\varphi_4, \varphi_6, \varphi_7)$ have uncertainty in their satisfaction as of the conflicting results over the two models. Finally, property (φ_5) is not satisfied in the model as of the similar result with value (F) obtained from the two two-valued models. To check the scalability of our approach,



```
 MODULE main
VAR
Env1 : Environment1 (Sys1.action,Us1.action,Motion1.action,DoorB1.action,SC1.action,WM1.action,FireA1.action,FireD1.action,Sm
Sys1 : System1 ( Us1.action,Env1.action,Motion1.action,DoorB1.action,SC1.action,WM1.action,FireA1.action,FireD1.action,SmogA
Motion1 : MotionS1 (Us1.action,Env1.action,Sys1.action,DoorB1.action,SC1.action,WM1.action,FireA1.action,FireD1.action,SmogA
DoorB1: DoorBell1 (Us1.action,Env1.action,Sys1.action,Motion1.action,SC1.action,WM1.action,FireA1.action,FireD1.action,SmogA
SC1 : Scam1 (Us1.action,Env1.action,Sys1.action,Motion1.action,DoorB1.action,WM1.action,FireA1.action,FireD1.action,SmogA1.a
WM1 : WashM1 (Us1.action,Env1.action,Sys1.action,Motion1.action,DoorB1.action,SC1.action,FireA1.action,FireD1.action,SmogA1
FireA1 : FireAlarm1 (Us1.action,Env1.action,Sys1.action,Motion1.action,DoorB1.action,SC1.action,WM1.action,FireD1.action,Smog
FireD1 : FireDep1 (Us1.action,Env1.action,Sys1.action,Motion1.action,DoorB1.action,SC1.action,WM1.action,FireA1.action,SmogA
SmogA1 : SmogAlarm1 (Us1.action,Env1.action, Sys1.action,Motion1.action,DoorB1.action,SC1.action,WM1.action,FireA1.action,F
Therm1 : Thermos1 (Us1.action,Env1.action,Sys1.action,Motion1.action,DoorB1.action,SC1.action,WM1.action,FireA1.action,FireD
Us1 : User1 (Env1.action,Sys1.action,Motion1.action,DoorB1.action,SC1.action,WM1.action,FireA1.action,FireD1.action,SmogA1.ac



DEFINE sysready := Env1.state = e3;
DEFINE motion := Env1.state = e0 ;
DEFINE campic := Env1.state = e2;
DEFINE frontlight =M := Env1.state = e1 | Env1.state = e2 | Env1.state = e0;
DEFINE doorbell=M :=  Env1.state = e2;
```

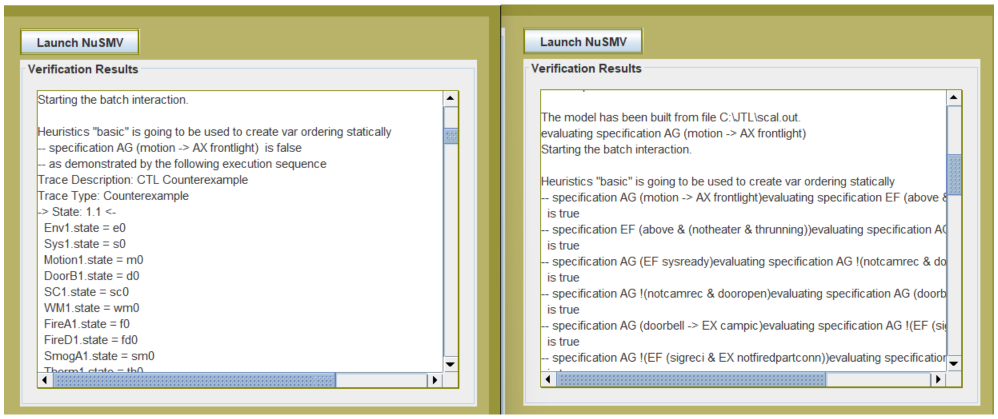
Figure 7.3: Uploading 3v-CTL



```
Motion1 : MotionS1 (Us1.action,Env1.action,Sys1.action,DoorB1.action,SC1.action,WM1.
DoorB1: DoorBell1 (Us1.action,Env1.action,Sys1.action,Motion1.action,SC1.action,WM1.
SC1 : Scam1 (Us1.action,Env1.action,Sys1.action,Motion1.action,DoorB1.action,WM1.ac
WM1 : WashM1 (Us1.action,Env1.action,Sys1.action,Motion1.action,DoorB1.action,SC1.a
FireA1 : FireAlarm1 (Us1.action,Env1.action,Sys1.action,Motion1.action,DoorB1.action,SC
FireD1 : FireDep1 (Us1.action,Env1.action,Sys1.action,Motion1.action,DoorB1.action,SC
SmogA1 : SmogAlarm1 (Us1.action,Env1.action, Sys1.action,Motion1.action,DoorB1.actio
Therm1 : Thermos1 (Us1.action,Env1.action,Sys1.action,Motion1.action,DoorB1.action,S
Us1 : User1 (Env1.action,Sys1.action,Motion1.action,DoorB1.action,SC1.action,WM1.acti



-- Atomic Propositions
DEFINE sysready := Env1.state = e3;
DEFINE motion := Env1.state = e0 ;
DEFINE campic := Env1.state = e2;
DEFINE frontlight :=FALSE;
DEFINE doorbell:=FALSE;
DEFINE remindsent :=Env1.state = e6;
DEFINE load :=FALSE;
DEFINE notload := Env1.state = e7;
DEFINE wmon:= Env1.state = e6;
```

```
Env1 : Environment1 (Sys1.action,Us1.action,Motion1.action,DoorB1.action,SC1.action,W
Sys1 : System1 ( Us1.action,Env1.action,Motion1.action,DoorB1.action,SC1.action,WM1.
Motion1 : MotionS1 (Us1.action,Env1.action,Sys1.action,DoorB1.action,SC1.action,WM1.
DoorB1: DoorBell1 (Us1.action,Env1.action,Sys1.action,Motion1.action,SC1.action,WM1.
SC1 : Scam1 (Us1.action,Env1.action,Sys1.action,Motion1.action,DoorB1.action,WM1.ac
WM1 : WashM1 (Us1.action,Env1.action,Sys1.action,Motion1.action,DoorB1.action,SC1.a
FireA1 : FireAlarm1 (Us1.action,Env1.action,Sys1.action,Motion1.action,DoorB1.action,SC
FireD1 : FireDep1 (Us1.action,Env1.action,Sys1.action,Motion1.action,DoorB1.action,SC
SmogA1 : SmogAlarm1 (Us1.action,Env1.action, Sys1.action,Motion1.action,DoorB1.actio
Therm1 : Thermos1 (Us1.action,Env1.action,Sys1.action,Motion1.action,DoorB1.action,S
Us1 : User1 (Env1.action,Sys1.action,Motion1.action,DoorB1.action,SC1.action,WM1.acti



-- Atomic Propositions
DEFINE sysready := Env1.state = e3;
DEFINE motion := Env1.state = e0 ;
DEFINE campic := Env1.state = e2;
DEFINE frontlight   := Env1.state = e1 | Env1.state = e2 | Env1.state = e0;
DEFINE doorbell  :=  Env1.state = e2;
DEFINE remindsent :=Env1.state = e6;
DEFINE load   := Env1.state = e7;
DEFINE notload := Env1.state = e7;
```

Figure 7.4: The positive and negative cuts derived from the 3v-CTL model



```
Starting the batch interaction.

Heuristics "basic" is going to be used to create var ordering statically
-- specification AG (motion -> AX frontlight)  is false
-- as demonstrated by the following execution sequence
Trace Description: CTL Counterexample
Trace Type: Counterexample
-> State: 1.1 <-
  Env1.state = e0
  Sys1.state = s0
  Motion1.state = m0
  DoorB1.state = d0
  SC1.state = sc0
  WM1.state = wm0
  FireA1.state = f0
  FireD1.state = fd0
  SmogA1.state = sm0
```

```
The model has been built from file C:\JTL\scal.out.
evaluating specification AG (motion -> AX frontlight)
Starting the batch interaction.

Heuristics "basic" is going to be used to create var ordering statically
-- specification AG (motion -> AX frontlight)evaluating specification EF (above &
 is true
-- specification EF (above & (notheater & thrunning))evaluating specification AG
 is true
-- specification AG (EF sysready)evaluating specification AG !(notcamrec & do
 is true
-- specification AG !(notcamrec & dooropen)evaluating specification AG (doorb
 is true
-- specification AG (doorbell -> EX campic)evaluating specification AG !(EF (si
 is true
-- specification AG !(EF (sigreci & EX notfiredpartconn))evaluating specificatio
```

Figure 7.5: The verification results of the positive and negative cuts

Table 7.1: The verification results of the Smart Home model

Pro.	**Pos.**	**Neg.**	**Result**
φ_1	T	T	T
φ_2	T	T	T
φ_3	T	T	T
φ_4	T	F	M
φ_5	F	F	F
φ_6	T	F	M
φ_7	T	F	M

Table 7.2: The results of scalability after running the tool over the positive and negative models ten times

Exp.#	Age.#	St.#	T.time(ms)	M.time(ms)
1	11	14	7.6	5.9
2	22	196	09.8	11.4
3	33	2744	16.2	12.2
4	44	38416	81.2	24.4
5	55	537824	35.9	29.2
6	66	7.52954e+006	49.2	40.8
7	77	1.05414e+008	77.11	65.03
8	88	1.47579e+009	110.44	124.90
9	99	2.0661e+010	132.36	126.97
10	110	2.89255e+011	377.09	323.26

we conducted ten experiments; the first started with 11, and the last ended with 110 agents. In each experiment, we added one instance for each agent, and by reaching the last experiment, we had 10 instances for each agent. Our tool ran on a machine with the following specifications: 12th Gen Intel(R) Core(TM) i5-1235U, with 1300 Mhz, 10Core and 12 Logical processors. Table 7.2 shows the logarithmic relationship between the number of agents (Age.#) and the verification time of the two models (T.time(ms)) and (M.time(ms)). Moreover, the table shows the exponential increase in the number of states (St.#) according to the increase in agents.

7.1.2 Case study 2: A 3v-CTL^{cc} Smart Hospital System with Uncertainty

We consider a specific scenario of a Smart Hospital system with multi-source data as an example. The scenario is described in [8], where most of the interactions between agents are based on conditional commitment protocols. The scenario is depicted in Figure 7.6. The figure illustrates a particular path or sequence of events rather than capturing the entire system's complexity. The scenario serves as an abstraction of the larger system. It highlights critical interactions and behaviors related to the commitment protocols while omitting some intricacies of the full system for clarity and conciseness. The scenario contains multiple agents: Nurse, System, Robot, Patient, Physician, and Sensors. These agents interact as follows: (1) the patient arrives at the hospital and automatically checks-in; (2) the GPS installed on the patient's smartphone commits to showing the directions to the reception room; (3) the system then commits to notifying the nurse to meet the patient, which in turn, commits to meeting the patient; (4) the system notifies the bed washer robot to disinfect and prepare the bed, and the robot commits to doing so; (5) after the diagnosis, the patient arrives at the room for hospitalization; (6) then, the smart sensors and devices connect to the patient's body and bed to read the required health information; (7) when the blood pressure sensor reads a high-level pressure, it sends an alert to the nurse and physician, and the latter commits to assigning a treatment for the patient; (8) the system commits to showing a list of meals to the patient, which then commits to reviewing and choosing the

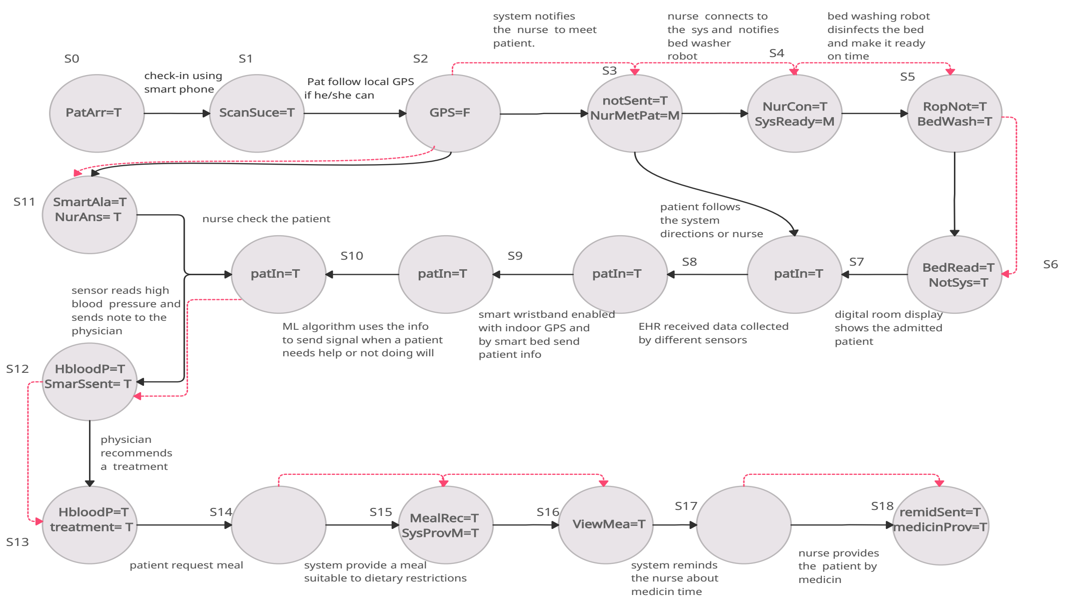

S0
PatArr=T
check-in using smart phone
S1
ScanSuce=T
Pat follow local GPS if he/she can
S2
GPS=F
system notifies the nurse to meet patient.
S3
notSent=T NurMetPat=M
nurse connects to the sys and notifies bed washer robot
S4
NurCon=T SysReady=M
bed washing robot disinfects the bed and make it ready on time
S5
RopNot=T BedWash=T
S6
BedRead=T NotSys=T
S7
patIn=T
digital room display shows the admitted patient
patient follows the system directions or nurse
S8
patIn=T
EHR received data collected by different sensors
S9
patIn=T
smart wristband enabled with indoor GPS and by smart bed send patient info
S10
patIn=T
ML algorithm uses the info to send signal when a patient needs help or not doing will
patIn=T
nurse check the patient
S11
SmartAla=T NurAns= T
sensor reads high blood pressure and sends note to the physician
S12
HbloodP=T SmarSsent= T
physician recommends a treatment
S13
HbloodP=T treatment= T
patient request meal
S14
S15
MealRec=T SysProvM=T
system provide a meal suitable to dietary restrictions
S16
ViewMea=T
S17
system reminds the nurse about medicin time
nurse provides the patient by medicin
S18
remidSent=T medicinProv=T

Figure 7.6: Example of a scenario in a Smart Hospital system

desired meal; and (9) the nurse commits to providing the patient with medicine when the system sends a reminder.

The system is modeled so that some atomic propositions in states are assigned with absolute (true) or (false). For example, “NurCon=T” stands for *it is true that the nurse is connected to the system* in $S3$. Some atomic propositions take the value (maybe), such as “SysReady=M”, which means *It is uncertain that the system is ready* in state s_4.

System Specifications

In this section, as in the previous case study, we verify ten properties, including *Safety*, *Liveness* and *Reachability* properties. In these properties, $n1$ stands for agent Nurse, $s1$ for System, $r1$ for Robot, $p1$ for Patient, $ph1$ for Physician and $sen1$ for agent Sensor.

- $\varphi_1 = \neg\, EF\ \ CC_{n1\to s1}(SmartAla, \neg NurAnswer)$ means there is no possible execution where the **system sends a signal through the smart alarm** to the nurse but the nurse commits to **not responding**.

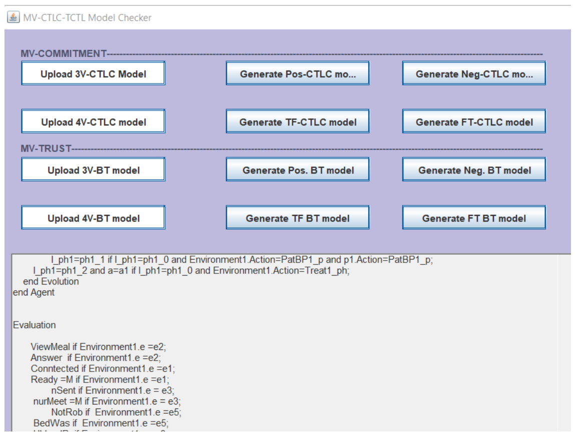
- $\varphi_2 = \neg\, EF\ CC_{r1\to s1}(\neg BedRead, NotSys)$ means there is no possible execution where

the bed is not ready, but the washer robot commits to **notifying the system** of the bed availability.

- $\varphi_3 = AF \ CC_{n1 \to p1}(remindSent, midicinProv)$ means when the system **sends a reminder** to the nurse, the latter commits to **providing medicine** to the patient.
- $\varphi_4 = AF \ CC_{s1 \to n1}(NurCon, SysReady)$ means when the nurse **needs to connect**, the system commits to be **ready**.
- $\varphi_5 = EF \ CC_{n1 \to s1}(notSent, NurMetPat)$ means when the system **sends a patent's arriving notification** to the nurse, the latter commits to **meeting the patient**.
- $\varphi_6 = EF \ CC_{r1 \to s1}(RobNot, BedWash)$ means when the system **notifies the robot** to prepare the bed, the robot commits to **washing and preparing the bed**.
- $\varphi_7 = EF \ CC_{ph1 \to p1}(HbloodP, treatment)$ means when the **patient has high blood pressure**, the physician commits to **recommend the required treatment**.
- $\varphi_8 = EF \ CC_{sen1 \to p1}(HbloodP, SmartSsent)$ means when the **patient has high blood pressure**, the sensor commits to **send alert to the nurse** to see the patient.
- $\varphi_9 = EF \ CC_{n1 \to s1}(SmartAla, NurAns)$ means when the system **sends an alert** to the nurse, the latter commits to **answering the alert call**.
- $\varphi_{10} = EF \ CC_{p1 \to s1}(viewMea, Answer)$ means when the system sends the **meal list to be viewed by the patient**, the latter commits to **answering** by choosing a meal or refusing.

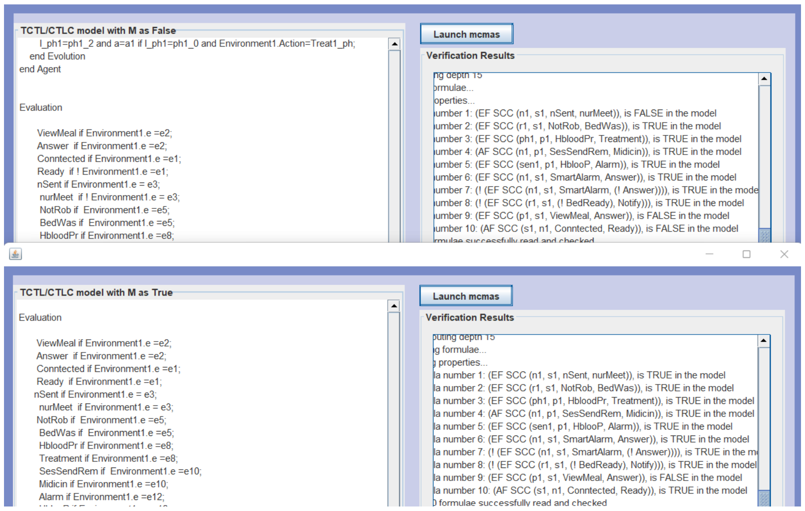
System Verification

We check the model by fully implementing our new reduction algorithm (Algorithm 2) using the new tool MV-Checker which automatically translates 3v-CTL^{cc} to the two-valued CTL^{cc} logic and automatically interacts with the **MCMAS+** model checker developed for multi-agent social commitments [31]. Figure 7.7 shows the first step of the implementation, which is the uploading of the 3v-CTL^{cc} model. Then, after pressing on "*Generate Positive*



```
         l_ph1=ph1_1 if l_ph1=ph1_0 and Environment1.Action=PatBP1_p and p1.Action=PatBP1_p;
      l_ph1=ph1_2 and a=a1 if l_ph1=ph1_0 and Environment1.Action=Treat1_ph;
   end Evolution
end Agent


Evaluation

     ViewMeal if Environment1.e =e2;
     Answer  if Environment1.e =e2;
     Conntected if Environment1.e =e1;
     Ready =M if Environment1.e =e1;
           nSent if Environment1.e = e3;
      nurMeet =M if Environment1.e = e3;
           NotRob if  Environment1.e =e5;
      BedWas if  Environment1.e =e5;
```

Figure 7.7: Uploading 3v-CTL^{cc} model



```
      l_ph1=ph1_2 and a=a1 if l_ph1=ph1_0 and Environment1.Action=Treat1_ph;
   end Evolution
end Agent


Evaluation

      ViewMeal if Environment1.e =e2;
      Answer  if Environment1.e =e2;
      Conntected if Environment1.e =e1;
      Ready  if ! Environment1.e =e1;
      nSent if Environment1.e = e3;
       nurMeet  if ! Environment1.e = e3;
       NotRob if  Environment1.e =e5;
       BedWas if  Environment1.e =e5;
       HbloodPr if Environment1.e =e8;
```

```
ng depth 15
ormulae...
operties...
umber 1: (EF SCC (n1, s1, nSent, nurMeet)), is FALSE in the model
umber 2: (EF SCC (r1, s1, NotRob, BedWas)), is TRUE in the model
umber 3: (EF SCC (ph1, p1, HbloodPr, Treatment)), is TRUE in the model
umber 4: (AF SCC (n1, p1, SesSendRem, Midicin)), is TRUE in the model
umber 5: (EF SCC (sen1, p1, HblooP, Alarm)), is TRUE in the model
umber 6: (EF SCC (n1, s1, SmartAlarm, Answer)), is TRUE in the model
umber 7: (! (EF SCC (n1, s1, SmartAlarm, (! Answer)))), is TRUE in the mode
umber 8: (! (EF SCC (r1, s1, (! BedReady), Notify))), is TRUE in the model
umber 9: (EF SCC (p1, s1, ViewMeal, Answer)), is FALSE in the model
umber 10: (AF SCC (s1, n1, Conntected, Ready)), is FALSE in the model
rmulae successfully read and checked
```

```
Evaluation

      ViewMeal if Environment1.e =e2;
      Answer  if Environment1.e =e2;
      Conntected if Environment1.e =e1;
      Ready  if Environment1.e =e1;
     nSent if Environment1.e = e3;
       nurMeet  if Environment1.e = e3;
      NotRob if  Environment1.e =e5;
       BedWas if  Environment1.e =e5;
       HbloodPr if Environment1.e =e8;
       Treatment if Environment1.e =e8;
       SesSendRem if  Environment1.e =e10;
       Midicin if Environment1.e =e10;
       Alarm if Environment1.e =e12;
```

```
puting depth 15
g formulae...
g properties...
la number 1: (EF SCC (n1, s1, nSent, nurMeet)), is TRUE in the model
la number 2: (EF SCC (r1, s1, NotRob, BedWas)), is TRUE in the model
la number 3: (EF SCC (ph1, p1, HbloodPr, Treatment)), is TRUE in the model
la number 4: (AF SCC (n1, p1, SesSendRem, Midicin)), is TRUE in the model
la number 5: (EF SCC (sen1, p1, HblooP, Alarm)), is TRUE in the model
la number 6: (EF SCC (n1, s1, SmartAlarm, Answer)), is TRUE in the model
la number 7: (! (EF SCC (n1, s1, SmartAlarm, (! Answer)))), is TRUE in the m
la number 8: (! (EF SCC (r1, s1, (! BedReady), Notify))), is TRUE in the model
la number 9: (EF SCC (p1, s1, ViewMeal, Answer)), is FALSE in the model
la number 10: (AF SCC (s1, n1, Conntected, Ready)), is TRUE in the model
0 formulae successfully read and checked
```

Figure 7.8: Verification results of the two-valued models derived from 3v-CTL^{cc}

CTLC" and "*Generate Negative CTLC*" buttons, we get the two two-valued CTL^{cc} models in Figure 7.8. The last step is to press "*Launch mcmas*" over each model to have the verification results.

Verification Results

We performed our experiments using the same machine in the previous section. Table 7.3 shows the obtained results: the safety properties φ_1 and φ_2 are satisfied, the reachability properties $\varphi_6, \varphi_7, \varphi_8$ and φ_9, and the liveness property φ_3, as well are also satisfied. However, the property φ_{10} is not satisfied in the model. Moreover, it is unknown whether properties φ_4 and φ_5 are satisfied because of the conflicting results. For example, in Figure 7.8, the property "$\varphi_5 = EF \ CC_{n1 \to s1}(notSent, NurMetPat)$" has the value M because the negative and the positive cuts yield different answers. We performed ten experiments to check the scalability of our proposed approach. The results are reported in Table 7.4. We started with seven agents and ended with 70. The table shows the number of agents (#Age), the number of reachable states (#States), the average of the verification times and the memory in use in megabytes for the two two-valued models. The results reflect that the number of reachable states increases exponentially with the increase in the number of agents, while the memory in use and the verification time increases logarithmically.

Table 7.3: The verification results of the Smart Hospital model

Pro.	**M.T**	**M.F**	**Result**
φ_1	T	T	T
φ_2	T	T	T
φ_3	T	T	T
φ_4	T	F	M
φ_5	T	F	M
φ_6	T	T	T
φ_7	T	T	T
φ_8	T	T	T
φ_9	T	T	T
φ_{10}	F	F	F

7.1.3 Case Study 3: A 3v-CTL^c Intelligent Mortgage System with Uncertainty

We perform our experiments with an Intelligent Contract Mortgage system scenario with multi-source data described in [3]. The system is intelligent because it has the following characteristics: (1) Data Classification: The scenario involves the handling of various types

Table 7.4: Scalability results of running the tool ten times, starting with seven and ending with 70 agents.

#Age.	**#St**	**Ave.(MB)**	**Ave.time(ms)**
7	174	11.937	0.024
14	2668	13.779	0.380
21	44856	18.409	1.427
28	779440	24.120	4.026
35	1.37423e+07	30.590	10.95
42	2.44159e+08	32.594	13.34
49	4.3578e+09	62.485	16.729
56	7.80032e+10	42.401	25.243
63	1.39886e+12	101.637	43.945
70	2.51172e+13	109.071	69.281

of data, including property valuation reports, validation reports, municipality reports, and title search reports. Each type of data is processed and classified according to its nature and purpose in the mortgage application process. (2) Smart Decisions: The use of smart contracts enables automated decision-making based on predefined conditions. For example, the smart contract checks the validation reports to determine if they meet the required criteria before proceeding. Similarly, the buyer's bank uses automated decision-making to approve the loan if all conditions are satisfied. These elements contribute to the intelligence of the mortgage system by incorporating automation, predefined criteria, and data processing into the workflow. This improves efficiency, accuracy, and transparency in the mortgage application process.

The system's functionalities are based on unconditional commitment protocols under uncertainty. The scenario starts when a buyer applies for property valuation to the mortgage solutions firm remotely through a smartphone. Then the property valuation firm commits to submitting the application to the broker firm for validation which, in turn, commits to validating the report via a smart contract. After validation, the broker firm commits

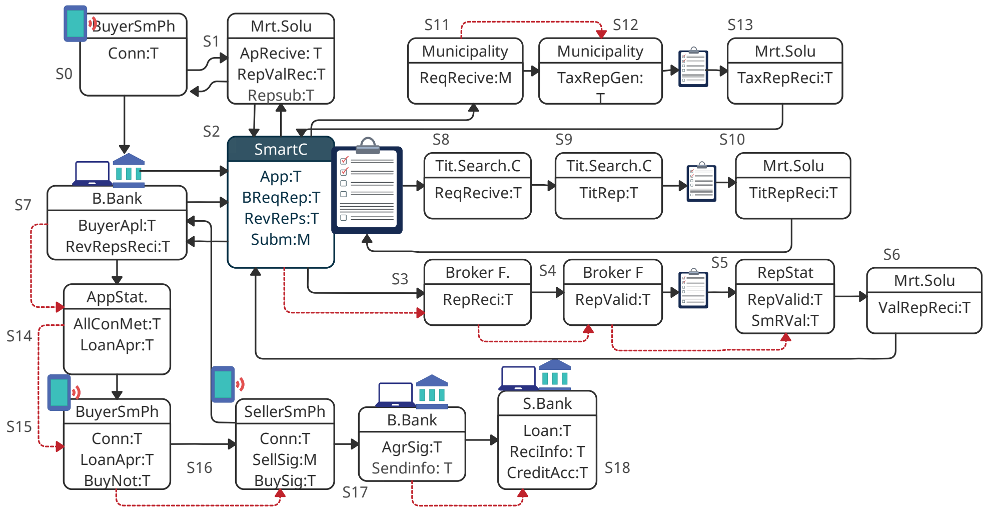


Figure 7.9: A Smart Mortgage system based on a smart contract

to resubmitting the report to the mortgage solutions firm. Then, the latter commits to approving the request and notifying the buyer. When a smart contract checks all the conditions, the buyer applies for a loan. At the same time, the buyer's bank requests a report generation from the municipality and title search. Next, the latter commits to generating and submitting their reports to the mortgage solutions firm. The smart contract then commits to checking the reports and forwarding them to the buyer's bank. The buyer's bank commits to approving the loan if all the conditions are satisfied. At the same time, the buyer's bank commits to automatically notifying the buyer of the new information. Thereafter, the seller and buyer commit to signing the agreement. After signing the agreement, the buyer's bank commits to updating the seller's bank with the new information through a smart contract. Figure 7.9 shows the system model where the red dashed lines indicate the commitment relations and the atomic propositions take truth values among T, F, and M.

System Specifications

We checked the following nine unconditional commitment properties over this system as follows:

- $\varphi_1 = \neg EF(\neg RepVal \ \wedge \ C_{SmCon_1 \to BFm_1}(SmRVal))$ means there is no possible execution where the broker firm **doesn't validate the submitted report**, and the smart contract commits to **updating the report's status to valid.**

- $\varphi_2 = \neg EF \ \neg C_{BBuy_1 \to Buyer_1}(BuyNot))$ means there is no possible execution where the buyer's bank doesn't commit to **sending a notification to the buyer** after each file update.

- $\varphi_3 = EF \ C_{SmCont_1 \to MortS_1}(Repsub)$ means the smart contract commits to **submitting the validated report** to the mortgage solution.

- $\varphi_4 = C_{Muni_1 \to SmCont_1}(TaxRepGen)$ means the municipality commits to **generating a tax report** after the generating request is sent by the smart contract.

- $\varphi_5 = EF \ C_{BBuy_1 \to BSell_1}(Sendinfo)$ means the Bank of the buyer commits to **sending information to the seller's bank**.

- $\varphi_6 = EF \ C_{BFm_1 \to SmCont_1}(RepValid)$ means the broker firm commits to **validating the submitted report**.

- $\varphi_7 = EF \ (BuyerApl \wedge AX(C_{MortS_1 \to Buyr_1}(Subm)))$ means after **the buyer applies for a mortgage**, the mortgage solution immediately commits to **submitting the report for validation**.

- $\varphi_8 = EF \ LoanApr \ \wedge \ AX(C_{Seller_1 \to BBuy_1}(SellSig))$ means after **the loan is approved**, the seller immediately commits to **signing the agreement with the buyer**.

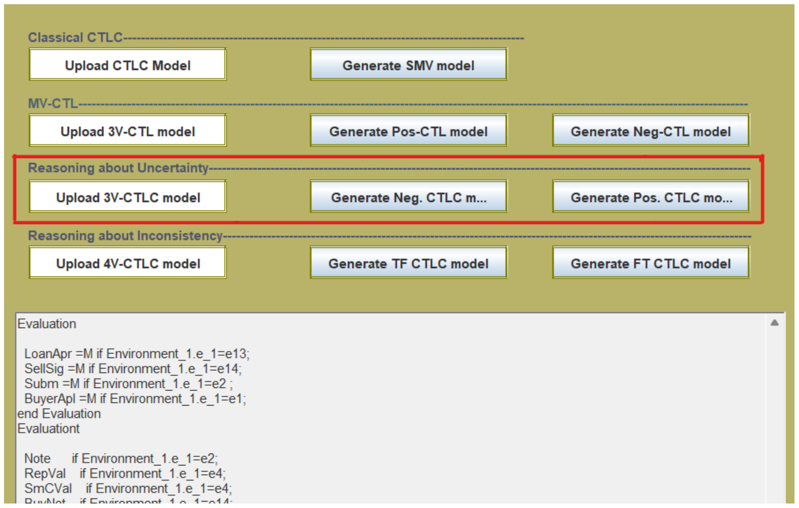
- $\varphi_9 = EF \ C_{SmCont_1 \to BFm_1}(SmRVal)$ means the smart contract commits to **submitting the application to the broker firm for validation**.

System Verification

We use the same machine in the previous sections for conducting our experiments. Figure 7.10 shows uploading the 3v-CTL^c model. Figure 7.11 shows the final results after generating the two-valued models and calling NuSMV over each model.

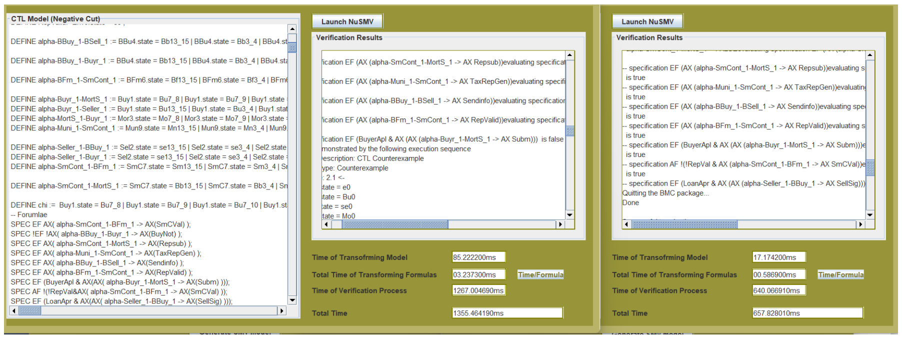
Verification Results

Table 7.5 shows the final results in column (Results) after comparing the verification results of the two-valued models M_m, which considers M as true and M_T, which considers only the value T as true. The results show that the properties φ_1, φ_3, φ_4, φ_5, φ_6 and φ_9 are satisfied. The φ_7 and φ_8 are uncertain if they are satisfied in the system, while φ_2 is not.



```
Evaluation

 LoanApr =M if Environment_1.e_1=e13;
 SellSig =M if Environment_1.e_1=e14;
 Subm =M if Environment_1.e_1=e2 ;
 BuyerApl =M if Environment_1.e_1=e1;
end Evaluation
Evaluationt

 Note      if Environment_1.e_1=e2;
 RepVal    if Environment_1.e_1=e4;
 SmCVal    if Environment_1.e_1=e4;
```

Figure 7.10: Uploading 3v-CTL^c model



```
DEFINE alpha-BBuy_1-BSell_1 := BBu4.state = Bb13_15 | BBu4.state = Bb3_4 | BBu4.st

DEFINE alpha-BBuy_1-Buyr_1 := BBu4.state = Bb13_15 | BBu4.state = Bb3_4 | BBu4.sta

DEFINE alpha-BFm_1-SmCont_1 := BFm6.state = Bf13_15 | BFm6.state = Bf3_4 | BFm6

DEFINE alpha-Buyr_1-MortS_1 := Buy1.state = Bu7_8 | Buy1.state = Bu7_9 | Buy1.state =
DEFINE alpha-Buyr_1-Seller_1 := Buy1.state = Bu13_15 | Buy1.state = Bu3_4 | Buy1.stat
DEFINE alpha-MortS_1-Buyr_1 := Mor3.state = Mo7_8 | Mor3.state = Mo7_9 | Mor3.state =
DEFINE alpha-Muni_1-SmCont_1 := Mun9.state = Mn13_15 | Mun9.state = Mn3_4 | Mun9.

DEFINE alpha-Seller_1-BBuy_1 := Sel2.state = se13_15 | Sel2.state = se3_4 | Sel2.state
DEFINE alpha-Seller_1-Buyr_1 := Sel2.state = se13_15 | Sel2.state = se3_4 | Sel2.state =
DEFINE alpha-SmCont_1-BFm_1 := SmC7.state = Sm13_15 | SmC7.state = Sm3_4 | Sr

DEFINE alpha-SmCont_1-MortS_1 := SmC7.state = Bb13_15 | SmC7.state = Bb3_4 | Sn

DEFINE chi :=  Buy1.state = Bu7_8 | Buy1.state = Bu7_9 | Buy1.state = Bu7_10 | Buy1.st
-- Forumlae
SPEC EF AX( alpha-SmCont_1-BFm_1 -> AX(SmCVal) );
SPEC !EF !AX( alpha-BBuy_1-Buyr_1 -> AX(BuyNot) );
SPEC EF AX( alpha-SmCont_1-MortS_1 -> AX(Repsub) );
SPEC EF AX( alpha-Muni_1-SmCont_1 -> AX(TaxRepGen) );
SPEC EF AX( alpha-BBuy_1-BSell_1 -> AX(Sendinfo) );
SPEC EF AX( alpha-BFm_1-SmCont_1 -> AX(RepValid) );
SPEC EF (BuyerApl & AX(AX( alpha-Buyr_1-MortS_1 -> AX(Subm) )));
SPEC AF !(!RepVal&AX( alpha-SmCont_1-BFm_1 -> AX(SmCVal) ));
SPEC EF (LoanApr & AX(AX( alpha-Seller_1-BBuy_1 -> AX(SellSig) )));
```

```
fication EF (AX (alpha-SmCont_1-MortS_1 -> AX Repsub))evaluating specificat

fication EF (AX (alpha-Muni_1-SmCont_1 -> AX TaxRepGen))evaluating specifi

fication EF (AX (alpha-BBuy_1-BSell_1 -> AX Sendinfo))evaluating specification

fication EF (AX (alpha-BFm_1-SmCont_1 -> AX RepValid))evaluating specificat

fication EF (BuyerApl & AX (AX (alpha-Buyr_1-MortS_1 -> AX Subm)))  is false
monstrated by the following execution sequence
escription: CTL Counterexample
ype: Counterexample
: 2.1 <-
tate = e0
tate = Bu0
ate = se0
tate = Mo0
```

Time of Transofrming Model: 85.222200ms
Total Time of Transforming Formulas: 03.237300ms
Time of Verification Process: 1267.004690ms
Total Time: 1355.464190ms

```
-- specification EF (AX (alpha-SmCont_1-MortS_1 -> AX Repsub))evaluating s
 is true
-- specification EF (AX (alpha-Muni_1-SmCont_1 -> AX TaxRepGen))evaluatin
 is true
-- specification EF (AX (alpha-BBuy_1-BSell_1 -> AX Sendinfo))evaluating spe
 is true
-- specification EF (AX (alpha-BFm_1-SmCont_1 -> AX RepValid))evaluating s
 is true
-- specification EF (BuyerApl & AX (AX (alpha-Buyr_1-MortS_1 -> AX Subm)))e
 is true
-- specification AF !(!RepVal & AX (alpha-SmCont_1-BFm_1 -> AX SmCVal))e
 is true
-- specification EF (LoanApr & AX (AX (alpha-Seller_1-BBuy_1 -> AX SellSig)))
Quitting the BMC package...
Done
```

Time of Transofrming Model: 17.174200ms
Total Time of Transforming Formulas: 00.586900ms
Time of Verification Process: 640.066910ms
Total Time: 657.828010ms

Figure 7.11: The verification results of the two models after calling NuSMV

To ensure the effectiveness of our approach, we ran our tool six times over multiple experiments. We started with ten and increased this number to 60 agents. Table 7.6 shows the results of our experiment where (#St.) and (#Age) represent the number of reachable states and the number of agents, respectively. The table shows the exponential increase in the number of reachable states according to the increase in the number of agents. As well as the average of the verification times (Verification(ms)) for the two models. The table also reports the average of the transformation times in milliseconds for transforming the two models (Models(ms)) and the formulae (formulae (ms)).

Table 7.5: The results of verifying the 3v-Smart Mortgage system

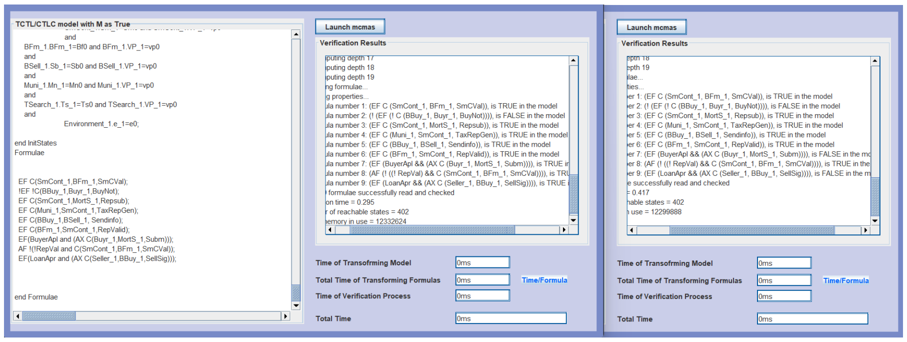
Pro.	M_m	M_T	**Result**
φ_1	T	T	T
φ_2	F	F	F
φ_3	T	T	T
φ_4	T	T	T
φ_5	T	T	T
φ_6	T	T	T
φ_7	T	F	M
φ_8	T	F	M
φ_9	T	T	T

7.1.4 Comparison with the Reduction Approach Using MCMAS+ and NuSMV

In this section, we implement our verification of the same Smart Mortgage system using our new algorithm (Algorithm 2), which translates 3v-CTL^c to CTL^c and then calls MCMAS+ to compare the new results the ones obtained using NuSMV in terms of scalability. Figure 7.12 shows the verification results after applying the reduction algorithm using MV-Checker interacting with MCMAS+. The verification results obtained using the first approach are

Table 7.6: Scalability results of running the MV-Checker tool six times over the two CTL models The times reported are the average of the models and formulae transformation times and the verification times.

#Age.	#St.	Models (ms)	Formulae (ms)	Verification(ms)
10	2226	10.622	1.153	1145
20	4955076	65.520	2.075	5224
30	1.103e+10	215.244	4.246	19801
40	2.45528e+13	640.358	4.901	406133
50	5.46545e+16	927.520	6.225	591204
60	1.21661e+20	1315.688	9.065	713551


TCTL/CTLC model with M as True

```
                    and
    BFm_1.BFm_1=Bf0 and BFm_1.VP_1=vp0
    and
    BSell_1.Sb_1=Sb0 and BSell_1.VP_1=vp0
    and
    Muni_1.Mn_1=Mn0 and Muni_1.VP_1=vp0
    and
    TSearch_1.Ts_1=Ts0 and TSearch_1.VP_1=vp0
    and
                    Environment_1.e_1=e0;

end InitStates
Formulae


 EF C(SmCont_1,BFm_1,SmCVal);
 !EF !C(BBuy_1,Buyr_1,BuyNot);
 EF C(SmCont_1,MortS_1,Repsub);
 EF C(Muni_1,SmCont_1,TaxRepGen);
 EF C(BBuy_1,BSell_1, Sendinfo);
 EF C(BFm_1,SmCont_1,RepValid);
 EF(BuyerApl and (AX C(Buyr_1,MortS_1,Subm)));
 AF !(!RepVal and C(SmCont_1,BFm_1,SmCVal));
 EF(LoanApr and (AX C(Seller_1,BBuy_1,SellSig)));



end Formulae
```

Launch mcmas

Verification Results

```
puting depth 17
puting depth 18
puting depth 19
ng formulae...
g properties...
ula number 1: (EF C (SmCont_1, BFm_1, SmCVal)), is TRUE in the model
ula number 2: (! (EF (! C (BBuy_1, Buyr_1, BuyNot)))), is FALSE in the model
ula number 3: (EF C (SmCont_1, MortS_1, Repsub)), is TRUE in the model
ula number 4: (EF C (Muni_1, SmCont_1, TaxRepGen)), is TRUE in the model
ula number 5: (EF C (BBuy_1, BSell_1, Sendinfo)), is TRUE in the model
ula number 6: (EF C (BFm_1, SmCont_1, RepValid)), is TRUE in the model
ula number 7: (EF (BuyerApl && (AX C (Buyr_1, MortS_1, Subm)))), is TRUE in
ula number 8: (AF (! ((! RepVal) && C (SmCont_1, BFm_1, SmCVal)))), is TRU
ula number 9: (EF (LoanApr && (AX C (Seller_1, BBuy_1, SellSig)))), is TRUE i
 formulae successfully read and checked
on time = 0.295
r of reachable states = 402
emory in use = 12332624
```

Time of Transofrming Model	0ms	
Total Time of Transforming Formulas	0ms	Time/Formula
Time of Verification Process	0ms	
Total Time	0ms	

Launch mcmas

Verification Results

```
epth 18
epth 19
lae...
ties...
er 1: (EF C (SmCont_1, BFm_1, SmCVal)), is TRUE in the model
er 2: (! (EF (! C (BBuy_1, Buyr_1, BuyNot)))), is FALSE in the model
er 3: (EF C (SmCont_1, MortS_1, Repsub)), is TRUE in the model
er 4: (EF C (Muni_1, SmCont_1, TaxRepGen)), is TRUE in the model
er 5: (EF C (BBuy_1, BSell_1, Sendinfo)), is TRUE in the model
er 6: (EF C (BFm_1, SmCont_1, RepValid)), is TRUE in the model
er 7: (EF (BuyerApl && (AX C (Buyr_1, MortS_1, Subm)))), is FALSE in the mo
er 8: (AF (! ((! RepVal) && C (SmCont_1, BFm_1, SmCVal)))), is TRUE in the
er 9: (EF (LoanApr && (AX C (Seller_1, BBuy_1, SellSig)))), is FALSE in the m
e successfully read and checked
= 0.417
hable states = 402
 use = 12299888
```

Time of Transofrming Model	0ms	
Total Time of Transforming Formulas	0ms	Time/Formula
Time of Verification Process	0ms	
Total Time	0ms	

Figure 7.12: The verification results of the two CTL^c models after calling MCMAS

the same as the ones obtained by interacting with NuSMV through the MV-Checker. Table 7.7 illustrates the results of the scalability to be compared with the results in Table 7.6. Figure 7.13 graphically shows the difference between the two algorithms regarding the number of agents and verification times. The brown color is assigned for the results of interacting MV-Checker with NuSMV, and the blue color is for the results of interacting with MCMAS+. The brown results show the exponential increase in the verification time with regard to the number of agents using the first tool. In contrast, the blue results show that the verification time increases logarithmically using the second tool. The letter tool enabled adding agents up to 90 agents, while the first tool reached up to 60 agents. Although both approaches gave reliable and accurate results, the graphical results reflect that, in terms of the verification time and the number of agents, the reduction approach based on using MCMAs+ required less time, and it is more scalable than the reduction one using NuSMV.

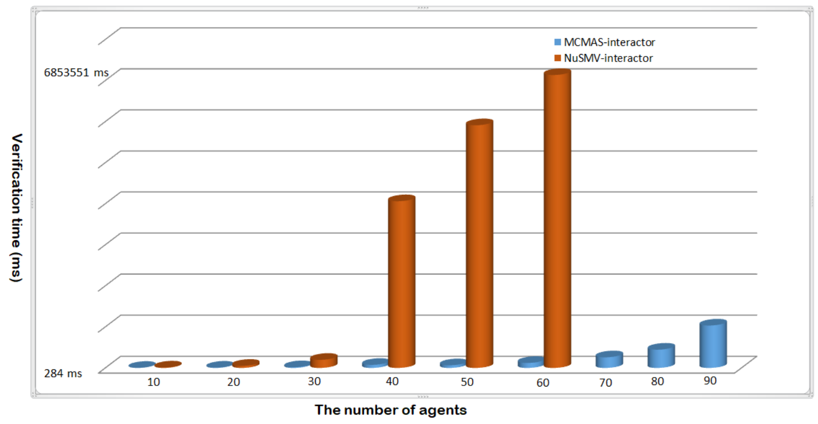


Figure 7.13: Comparison between the two transformation methods of $3v - CTL^c$, the blue results for CTL^c and the brown for CTL.

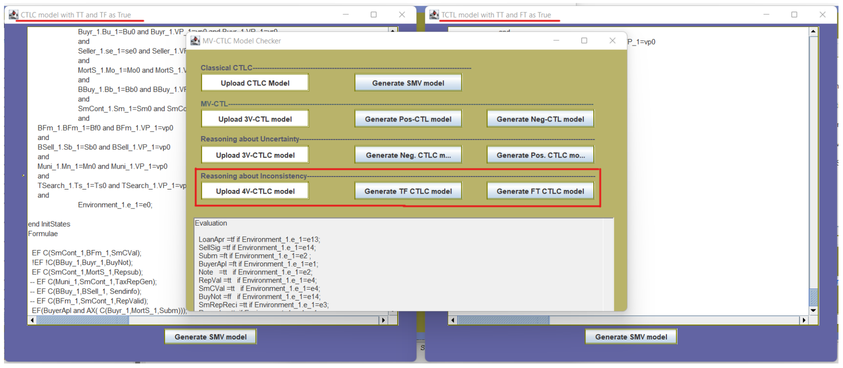
Table 7.7: The scalability results of running the tool over two two-valued CTL^c models

#Age.	**#States**	**Ave.(MB)**	**Ave.Time(ms)**
10	402	12.316	284
20	147606	22.479	764
30	5.66248e+07	57.248	1850
40	2.17433e+10	28.426	7531
50	8.34942e+12	33.142	6579
60	3.20618e+15	38.054	12209
70	1.23117e+18	49.890	25974
80	4.7277e+20	59.201	44547
90	7.3272e+21	75.013	103357

7.2 Modeling and Verifying Commitment Systems with Inconsistency

7.2.1 Case Study 4: A 4v-CTL^c Smart Mortgage System with Inconsistency

We use the same Smart Mortgage system with changing the variable truth values in the states to take truth values between *(TT, FF, FT or TF)*. We assume that the system is designed by two experts with different viewpoints about the system's atomic propositions. For example, the atomic propositions *SellSig =TF*, *BuyerApl=FT*, *SmCVal=TT* and *BuyNot=FF*. We apply our new reduction approach to transform the 4v-CTL^c model to CTL^c by considering the join-irreducible elements of the 4v-lattice (TF and FT). We verify the system against the properties (φ_1, φ_2, φ_3, φ_7, φ_8 and φ_9 assigned for the system in the previous section. From our results, we find that there are positive agreements (both designers say yes) about the satisfaction of the formulae φ_1, φ_3 and φ_9 and also a negative agreement (both designers say no) about the formula φ_2. The result FT shows the disagreement with information about "who said what" for the formula φ_7 where the first designer said "no" while the second said "yes". The opposite applies to φ_8 where the final result is TF. The obtained results are as expected, and we conclude that 4v-CTL^c is highly applicable for verifying IoTs and IS systems in terms of capturing the agreement and disagreement between the designers bout the system's behaviors.

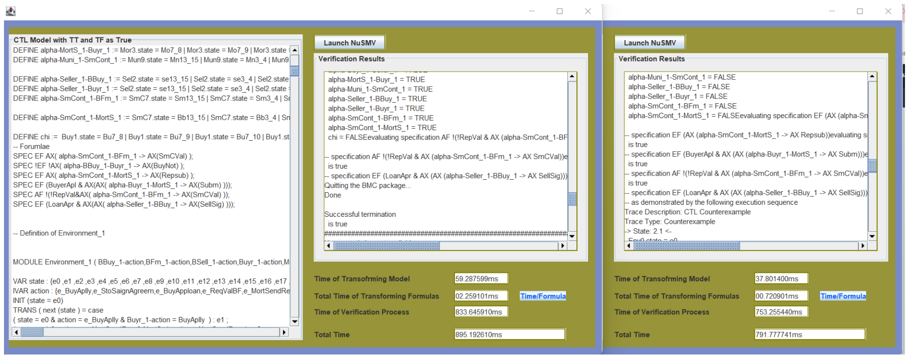


```
          Buyr_1.Bu_1=Bu0 and Buyr_1.VP_
          and
          Seller_1.se_1=se0 and Seller_1.VF
          and
          MortS_1.Mo_1=Mo0 and MortS_1.V
          and
          BBuy_1.Bb_1=Bb0 and BBuy_1.VF
          and
          SmCont_1.Sm_1=Sm0 and SmCo
          and
  BFm_1.BFm_1=Bf0 and BFm_1.VP_1=vp0
  and
  BSell_1.Sb_1=Sb0 and BSell_1.VP_1=vp0
  and
  Muni_1.Mn_1=Mn0 and Muni_1.VP_1=vp0
  and
  TSearch_1.Ts_1=Ts0 and TSearch_1.VP_1=vp
  and
          Environment_1.e_1=e0;

end InitStates
Formulae

 EF C(SmCont_1,BFm_1,SmCVal);
 !EF !C(BBuy_1,Buyr_1,BuyNot);
 EF C(SmCont_1,MortS_1,Repsub);
-- EF C(Muni_1,SmCont_1,TaxRepGen);
-- EF C(BBuy_1,BSell_1, Sendinfo);
-- EF C(BFm_1,SmCont_1,RepValid);
 EF(BuyerApl and AX( C(Buyr_1,MortS_1,Subm)));
```

```
Evaluation

 LoanApr =tf if Environment_1.e_1=e13;
 SellSig =tf if Environment_1.e_1=e14;
 Subm =ft if Environment_1.e_1=e2 ;
 BuyerApl =ft if Environment_1.e_1=e1;
 Note   =tt   if Environment_1.e_1=e2;
 RepVal =tt   if Environment_1.e_1=e4;
 SmCVal =tt   if Environment_1.e_1=e4;
 BuyNot =ff   if Environment_1.e_1=e14;
 SmRepReci =tt if Environment_1.e_1=e3;
```

```
P_1=vp0
```

Figure 7.14: Upload the 4v-CTL^c and generate the two-valued models



```
DEFINE alpha-MortS_1-Buyr_1 := Mor3.state = Mo7_8 | Mor3.state = Mo7_9 | Mor3.state
DEFINE alpha-Muni_1-SmCont_1 := Mun9.state = Mn13_15 | Mun9.state = Mn3_4 | Mun9

DEFINE alpha-Seller_1-BBuy_1 := Sel2.state = se13_15 | Sel2.state = se3_4 | Sel2.state
DEFINE alpha-Seller_1-Buyr_1 := Sel2.state = se13_15 | Sel2.state = se3_4 | Sel2.state
DEFINE alpha-SmCont_1-BFm_1 := SmC7.state = Sm13_15 | SmC7.state = Sm3_4 | S

DEFINE alpha-SmCont_1-MortS_1 := SmC7.state = Bb13_15 | SmC7.state = Bb3_4 | Sn

DEFINE chi :=  Buy1.state = Bu7_8 | Buy1.state = Bu7_9 | Buy1.state = Bu7_10 | Buy1.st
-- Forumlae
SPEC EF AX( alpha-SmCont_1-BFm_1 -> AX(SmCVal) );
SPEC !EF !AX( alpha-BBuy_1-Buyr_1 -> AX(BuyNot) );
SPEC EF AX( alpha-SmCont_1-MortS_1 -> AX(Repsub) );
SPEC EF (BuyerApl & AX(AX( alpha-Buyr_1-MortS_1 -> AX(Subm) )));
SPEC AF !(!RepVal&AX( alpha-SmCont_1-BFm_1 -> AX(SmCVal) ));
SPEC EF (LoanApr & AX(AX( alpha-Seller_1-BBuy_1 -> AX(SellSig) )));


-- Definition of Environment_1


MODULE Environment_1 ( BBuy_1-action,BFm_1-action,BSell_1-action,Buyr_1-action,M

VAR state : {e0 ,e1 ,e2 ,e3 ,e4 ,e5 ,e6 ,e7 ,e8 ,e9 ,e10 ,e11 ,e12 ,e13 ,e14 ,e15 ,e16 ,e17
IVAR action : {e_BuyAplly,e_StoSaignAgreem,e_BuyApploan,e_ReqValBF,e_MortSendRe
INIT (state = e0)
TRANS ( next (state ) = case
( state = e0 & action = e_BuyAplly & Buyr_1-action = BuyAplly  ) : e1 ;
```

```
 alpha-MortS_1-Buyr_1 = TRUE
 alpha-Muni_1-SmCont_1 = TRUE
 alpha-Seller_1-BBuy_1 = TRUE
 alpha-Seller_1-Buyr_1 = TRUE
 alpha-SmCont_1-BFm_1 = TRUE
 alpha-SmCont_1-MortS_1 = TRUE
 chi = FALSEevaluating specification AF !(!RepVal & AX (alpha-SmCont_1-BF

-- specification AF !(!RepVal & AX (alpha-SmCont_1-BFm_1 -> AX SmCVal))e
 is true
-- specification EF (LoanApr & AX (AX (alpha-Seller_1-BBuy_1 -> AX SellSig)))
Quitting the BMC package...
Done

Successful termination
 is true
################################################################
```

```
Time of Transofrming Model              59.287599ms
Total Time of Transforming Formulas     02.259101ms
Time of Verification Process            833.645910ms

Total Time                              895.192610ms
```

```
 alpha-Muni_1-SmCont_1 = FALSE
 alpha-Seller_1-BBuy_1 = FALSE
 alpha-Seller_1-Buyr_1 = FALSE
 alpha-SmCont_1-BFm_1 = FALSE
 alpha-SmCont_1-MortS_1 = FALSEevaluating specification EF (AX (alpha-Sn

-- specification EF (AX (alpha-SmCont_1-MortS_1 -> AX Repsub))evaluating s
 is true
-- specification EF (BuyerApl & AX (AX (alpha-Buyr_1-MortS_1 -> AX Subm)))e
 is true
-- specification AF !(!RepVal & AX (alpha-SmCont_1-BFm_1 -> AX SmCVal))e
 is true
-- specification EF (LoanApr & AX (AX (alpha-Seller_1-BBuy_1 -> AX SellSig)))
-- as demonstrated by the following execution sequence
Trace Description: CTL Counterexample
Trace Type: Counterexample
-> State: 2.1 <-
```

```
Time of Transofrming Model              37.801400ms
Total Time of Transforming Formulas     00.720901ms
Time of Verification Process            753.255440ms

Total Time                              791.777741ms
```

Figure 7.15: The two-valued CTL^c models and the results after launching NuSMV

It is worth clarifying that, in terms of the comparison with other related existing research, the work provided in this research is the first to extend CTL-based commitment logics to multi-valued versions and address the model checking problem of these logics. Therefore, our approaches cannot be compared with those related to other studies in similar frameworks. Moreover, the addressed applications used in this chapter are modeled and verified for the first time within the framework of commitment protocols with uncertainty or inconsistency using our logics. This led to several significant advantages, including the flexibility in describing their actual behaviours using multiple truth values and capturing

Table 7.8: The results of verifying the 4v-Smart Mortgage system

Pro.	**(TT-TF)**	**(TT-FT)**	**Result**
φ_1	T	T	TT
φ_2	F	F	FF
φ_3	T	T	TT
φ_7	F	T	FT
φ_8	T	F	TF
φ_9	T	T	TT

the commitment protocol, an essential communication protocol used in these applications. Furthermore, our work provides practical approaches for ensuring the reliability of the addressed applications.

7.3 Modeling and Verifying Trust Systems with Uncertainty

In this section, we multi-valued model check IS systems focusing on blockchain-based healthcare systems where trust occurs among the system's agents during their interactions. The system is considered intelligent based on the following characteristics: (1) Agent Collaboration: The system involves multiple intelligent agents (manufacturers, FDA, IPFS, smart contracts, distributors, patients, and pharmacies) working together to carry out a complex process. This collaborative effort reflects a form of artificial intelligence where agents interact based on predefined rules and conditions. (2)Smart Contracts and Blockchain Technology: The use of smart contracts and the IPFS (Interplanetary File System) demonstrates the application of advanced technology in the system. Smart contracts enable self-executing agreements based on predefined conditions, while the IPFS provides a decentralized and secure way to store and access data. Overall, the intelligence of the system lies in its ability to autonomously manage a complex process involving multiple stakeholders, make decisions based on predefined conditions, adapt to uncertainty, and utilize advanced technologies for secure and efficient operations.

7.3.1 Case Study 5: A 3v-TCTL Blockchain-Based Drug Traceability System under Uncertainty

The fifth case study is A blockchain-based system for drug traceability presented in [64]. We modelled the system as shown in Figure 7.16, where trust relations between agents are captured by Red-dashed lines and the uncertainty about holding some atomic propositions in states represented by value M.

System Functionalities

Seven primary agents operate within the system: (1) manufacturers; (2) FDA, which stands for the Food and Drug Administration; (3) IPFS, which stands for the Interplanetary File System; (4) smart contracts; (5) distributors; (6) patients; and (7) pharmacies. These agents interact with each other to carry out the system functions as follows. The system starts in state $S0$. In $S1$, it is (True) that a manufacturer does submit a request to the FDA for approval before commencing the drug Lot manufacturing. In $S2$, it is (True) that the FDA approves the valid request. In $S3$, it is (True) that the manufacturer initiates the manufacturing process. $S4$ shows that it is (True) that images of the drug Lot are uploaded to the IPFS by the manufacturer. In $S5$, it is (true) that the IPFS provides a hash saved in the smart contract, which allows authorized participants to access the images. It is (Uncertain) in $S6$ whether the drug is submitted to the distributor and whether a hash has already been sent. In $S7$, it is (True) that the distributor starts the drug distribution process, while in the same state, it is (Uncertain) that the drug is delivered simultaneously. In $S8$, it is (True) that the distributor uploads the package image to the IPFS. In $S9$, it is (True) that the IPFS sends a hash to the smart contract. as well as in $S10$, it is (True) that the drug packages are delivered to pharmacies. It is also (True) in $S11$ that the pharmacy begins selling the drug, and all supply chain participants be notified accordingly. In $S12$, it is (Uncertain) whether the image of the sold drug is uploaded by the pharmacy to the IPFS. In $S13$, it is (true) that the IPFS sends a hash to the smart contract. It is (True) in

$S14$ that the patient requests drugs, and in $S15$, the drug will be sold to the patient. To simplify the system, We eliminate the atomic propositions with the value False. The red dashed lines represent the trust accessibility relations between the system agents over the global system states.

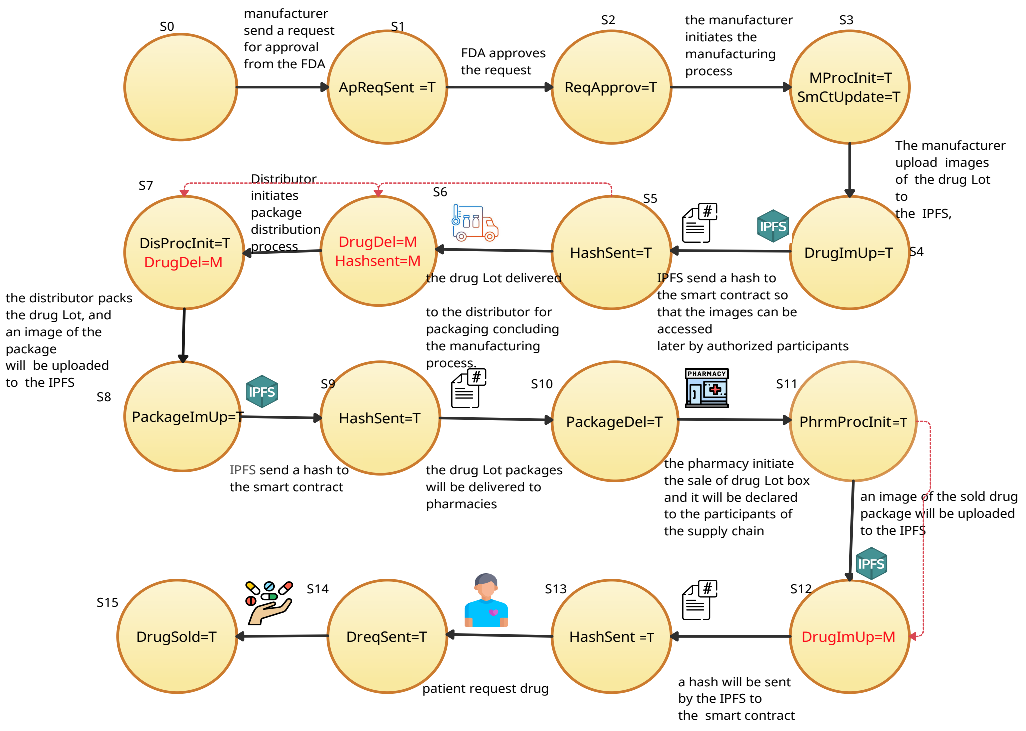

S0
manufacturer send a request for approval from the FDA
S1
ApReqSent =T
FDA approves the request
S2
ReqApprov=T
the manufacturer initiates the manufacturing process
S3
MProcInit=T
SmCtUpdate=T
The manufacturer upload images of the drug Lot to the IPFS,
S4
DrugImUp=T
IPFS
IPFS send a hash to the smart contract so that the images can be accessed later by authorized participants
S5
HashSent=T
S6
DrugDel=M
Hashsent=M
the drug Lot delivered to the distributor for packaging concluding the manufacturing process.
S7
DisProcInit=T
DrugDel=M
Distributor initiates package distribution process
the distributor packs the drug Lot, and an image of the package will be uploaded to the IPFS
S8
PackageImUp=T
IPFS
IPFS send a hash to the smart contract
S9
HashSent=T
the drug Lot packages will be delivered to pharmacies
S10
PackageDel=T
PHARMACY
the pharmacy initiate the sale of drug Lot box and it will be declared to the participants of the supply chain
S11
PhrmProcInit=T
an image of the sold drug package will be uploaded to the IPFS
IPFS
S12
DrugImUp=M
a hash will be sent by the IPFS to the smart contract
S13
HashSent =T
patient request drug
S14
DreqSent=T
S15
DrugSold=T

Figure 7.16: A Blockchain-Based Drug Traceability model with 3v-TCTL under uncertainty

System Specifications

This section focuses on the properties expressed in the temporal trust logic. We have verified five properties over this system, which are categorized into three categories: *Safety* properties convey the assurance that no undesirable outcomes will occur during the execution of the system. *Liveness* pertains to the eventual occurrence of favorable events, while *Reachability* refers to the achievement of a predetermined state within the system. We interpreted these properties in TCTL formal language and assigned atomic propositions as follows: "*ReqValid*" means " the request submitted by the manufacturer is valid ", "*ReqApprov*" means " FDA approve the request", "*DrugImUp*" means " drug package image is uploaded", "*HashSent*" means "the IPFS updates the smart contract by sending a hash", "*DrugDel*" means " the drug package is delivered ", "*DrugDela*" means "the drug package delivering time will be delayed". Below are the system properties :

(1) *"In every conceivable scenario, it is false that the request is valid and the manufacturer lacks trust in the FDA to approve the request for initiating the manufacturing process."*

$$\varphi_1 = \neg\ AF\ (ReqValid\ \ and\ \ \neg\ (T((Manuf,\ FDA,\ EF\ ReqApprov)))$$

(2) *" Each time an image of a drug package is uploaded, the manufacturer trusts the IPFS to update the smart contract using a hash".*

$$\varphi_2 = AG\ (DrugImUp\ and\ AX\ (T(Manuf,IPFS,\ T,\ HashSent)))$$

(3) *" The existence of at least one possible execution guarantees that during the drug delivery process, the IPFS trusts the manufacturer to upload the image of the drug package. "*

$$\varphi_3 = EF\ (DrugDel\ and\ (EF\ \ T(IPFS,\ Manuf,\ EF\ DrugImUp)))$$

(4) *"There exists at least one possible execution where the manufacturer trusts the IPFS to update the smart contract by a hash "*

$$\varphi_4 = EF\ \ T(Manuf,\ IPFS,\ HashSent).$$

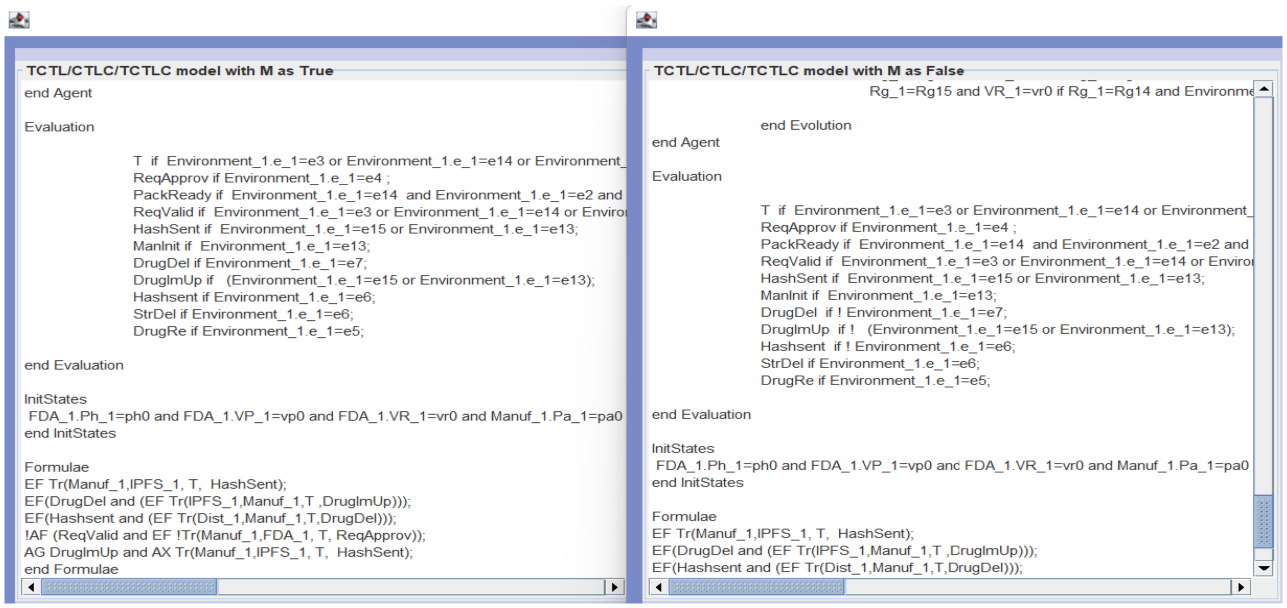
(5) *"There at least one possible execution ensures that, when a hash sent, the distributor trusts the manufacturer to send a notification in case of a delay in the drug package delivering"*

$$\varphi_5 = EF\ (HashSent\ and\ (EF\ T(Dist,\ Manuf,EF\ DrugDela)))$$

System Verification

After modelling the system in Figure 7.16, we encoded this model into 3v-VISPL, presented a set of specifications to be verified over this model, and saved the file. Then, we run MV-Checker to verify the system following the next steps. (1) Upload the 3v-VISPL by pressing the " Upload 3v-TCTL model" button on the main screen, as shown in Figure 6.7. (2) Transform the uploaded model into two classical TCTL models. The "Generate Pos. TCTL model" button generates the model that considers M and T as T. The " Generate Neg. TCTL model" button generates the model that considers M and F as F (Figure 7.17). (3) Call MCMAS^t over each model to get the verification results. Figure 7.18 shows that the formulae φ_1 and φ_4 are true in both models. While the formula φ_2 is false. The two formulae φ_3 and φ_5 are verified with different answers. Finally, we determine the final results based on the following approximation.

- Call MCMAS^t on the first model and save the result in x
- Call MCMAS^t on the second model and save the result in y
    - If $x = True$ and $y = True$ then, the final result: $T \sqcup M = T$
    - If $x = False$ and $y = True$ then, the final result: $\emptyset \sqcup M = M$
    - If $x = True$ and $y = False$ then, the final result: $T \sqcup \emptyset = T$
    - If $x = False$ and $y = False$ then, the final result: $\emptyset \sqcup \emptyset = \emptyset = F$

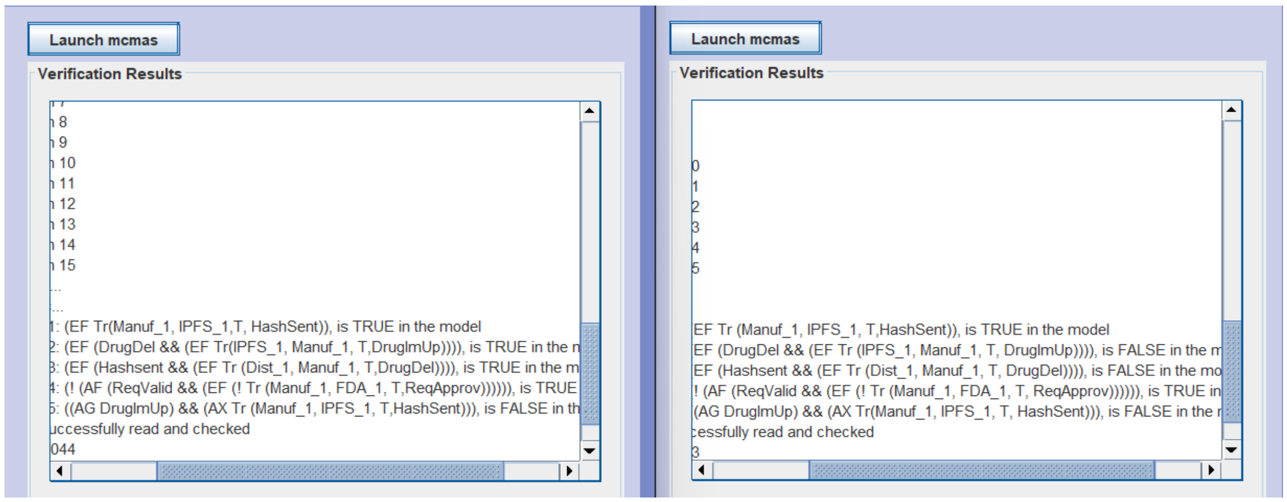

TCTL/CTLC/TCTLC model with M as True

```
end Agent

Evaluation

                    T  if  Environment_1.e_1=e3 or Environment_1.e_1=e14 or Environment_
                    ReqApprov if Environment_1.e_1=e4 ;
                    PackReady if  Environment_1.e_1=e14  and Environment_1.e_1=e2 and
                    ReqValid if  Environment_1.e_1=e3 or Environment_1.e_1=e14 or Enviro
                    HashSent if  Environment_1.e_1=e15 or Environment_1.e_1=e13;
                    ManInit if  Environment_1.e_1=e13;
                    DrugDel if Environment_1.e_1=e7;
                    DrugImUp if   (Environment_1.e_1=e15 or Environment_1.e_1=e13);
                    Hashsent if Environment_1.e_1=e6;
                    StrDel if Environment_1.e_1=e6;
                    DrugRe if Environment_1.e_1=e5;

end Evaluation

InitStates
 FDA_1.Ph_1=ph0 and FDA_1.VP_1=vp0 and FDA_1.VR_1=vr0 and Manuf_1.Pa_1=pa0
end InitStates

Formulae
EF Tr(Manuf_1,IPFS_1, T,  HashSent);
EF(DrugDel and (EF Tr(IPFS_1,Manuf_1,T ,DrugImUp)));
EF(Hashsent and (EF Tr(Dist_1,Manuf_1,T,DrugDel)));
!AF (ReqValid and EF !Tr(Manuf_1,FDA_1, T, ReqApprov));
AG DrugImUp and AX Tr(Manuf_1,IPFS_1, T,  HashSent);
end Formulae
```

TCTL/CTLC/TCTLC model with M as False

```
                                        Rg_1=Rg15 and VR_1=vr0 if Rg_1=Rg14 and Environme

                    end Evolution
end Agent

Evaluation

                    T  if  Environment_1.e_1=e3 or Environment_1.e_1=e14 or Environment_
                    ReqApprov if Environment_1.e_1=e4 ;
                    PackReady if  Environment_1.e_1=e14  and Environment_1.e_1=e2 and
                    ReqValid if  Environment_1.e_1=e3 or Environment_1.e_1=e14 or Enviro
                    HashSent if  Environment_1.e_1=e15 or Environment_1.e_1=e13;
                    ManInit if  Environment_1.e_1=e13;
                    DrugDel  if ! Environment_1.e_1=e7;
                    DrugImUp  if !   (Environment_1.e_1=e15 or Environment_1.e_1=e13);
                    Hashsent  if ! Environment_1.e_1=e6;
                    StrDel if Environment_1.e_1=e6;
                    DrugRe if Environment_1.e_1=e5;

end Evaluation

InitStates
 FDA_1.Ph_1=ph0 and FDA_1.VP_1=vp0 and FDA_1.VR_1=vr0 and Manuf_1.Pa_1=pa0
end InitStates

Formulae
EF Tr(Manuf_1,IPFS_1, T,  HashSent);
EF(DrugDel and (EF Tr(IPFS_1,Manuf_1,T ,DrugImUp)));
EF(Hashsent and (EF Tr(Dist_1,Manuf_1,T,DrugDel)));
```

Figure 7.17: The two classical TCTL obtained from the transformation of the 3v-TCTL



```
n 8
n 9
n 10
n 11
n 12
n 13
n 14
n 15
...
...
1: (EF Tr(Manuf_1, IPFS_1,T, HashSent)), is TRUE in the model
2: (EF (DrugDel && (EF Tr(IPFS_1, Manuf_1, T,DrugImUp)))), is TRUE in the n
3: (EF (Hashsent && (EF Tr (Dist_1, Manuf_1, T,DrugDel)))), is TRUE in the m
4: (! (AF (ReqValid && (EF (! Tr (Manuf_1, FDA_1, T,ReqApprov)))))), is TRUE
5: ((AG DrugImUp) && (AX Tr (Manuf_1, IPFS_1, T,HashSent))), is FALSE in th
uccessfully read and checked
044
```

```
0
1
2
3
4
5


EF Tr (Manuf_1, IPFS_1, T,HashSent)), is TRUE in the model
EF (DrugDel && (EF Tr (IPFS_1, Manuf_1, T, DrugImUp)))), is FALSE in the m
EF (Hashsent && (EF Tr (Dist_1, Manuf_1, T, DrugDel)))), is FALSE in the mo
! (AF (ReqValid && (EF (! Tr (Manuf_1, FDA_1, T, ReqApprov)))))), is TRUE in
(AG DrugImUp) && (AX Tr(Manuf_1, IPFS_1, T, HashSent))), is FALSE in the r
cessfully read and checked
3
```

Figure 7.18: The verification results over the two transformed classical TCTL models

The final verification results are illustrated in table 7.9. The table shows that the formulae φ_1 and φ_4 are true in the 3v-TCTL model, and the formula φ_2 is false, while the formulae φ_3 and φ_5 are verified with uncertainty. To check the scalability of the proposed approach, we increase the number of agents and run the tool over the 3v model. Table 7.10 presents the results of conducting our experiments. The table presents how many agents are included in each experiment (#Age), the number of the reachable states (#St), the average of the memory in use in megabytes taken from running the tool over the two generated classical models (Ave.(MB)), and the average of the verification time of each classical model (Ave.time(ms)). The results demonstrate that the number of reachable states increases exponentially with the number of agents, whereas the verification time increases logarithmically. These findings serve as evidence of the scalability of the proposed approach. We perform Our experiments over a machine with the following features: Intel(R) Core(TM) i7-6500U CPU 2.5GHz, 2Cors, 4 logical processors with 16GB Ram.

Table 7.9: Verification results of the Smart Drug Traceability system

Pro.	**Pos.**	**Neg.**	**Result**
φ_1	T	T	T
φ_2	F	F	F
φ_3	T	F	M
φ_4	T	T	T
φ_5	T	F	M

Table 7.10: Results of scalability after running the tool over the positive and negative models ten times

#Age.	**#St**	**Ave.(MB)**	**Ave.time(ms)**
7	20	9.4602	0.023
14	304	14.6711	1.31
21	4976	33.9584	3.19
28	83776	46.5274	6.83
35	1.42088e+06	49.7757	5.24
42	2.41417e+07	40.5451	20.70
49	4.10355e+08	37.9456	33.55
56	6.97582e+09	204.4452	48.01
63	1.18588e+11	46.75216	58.68
70	2.01599e+12	51.981296	65.77

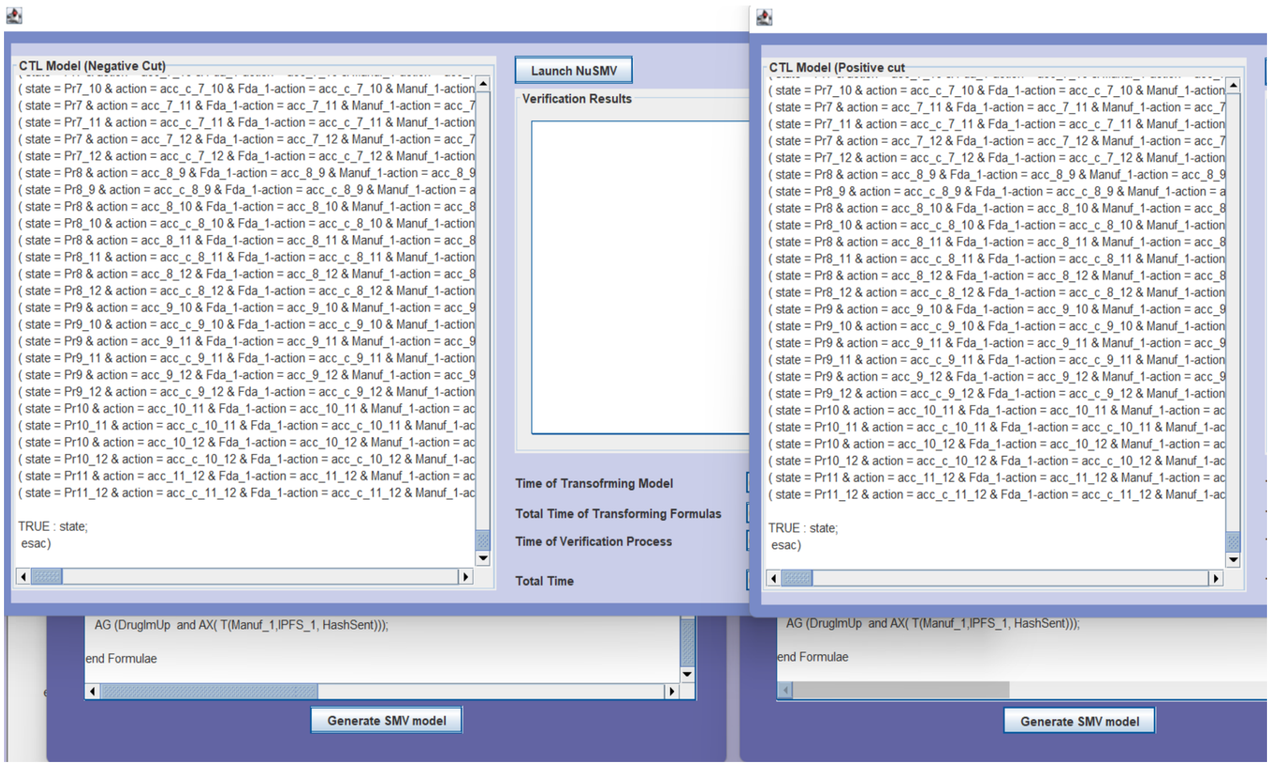
7.3.2 Case study 6: Re-Verifying the Blockchain-Based Drug Traceability System by Launching NuSMV

Here, we perform the second part of the MV-Checker's functions: launching an essential model checker, NuSMV.

As known, NuSMV is effectively used in verifying CTL-based models. In this section, we aim to investigate this tool's ability to verify the multi-valued versions of TCTL and CTLC logics. To do so, we implemented the reduction algorithms that transform the 3v-TCTL to CTL (Algorithm 7).

The running steps are as follows. First, press the " MV-TCTL to CTL" button from the main screen. Second, the second main frame for mv-trust appears with options to transfer the 3v-TCTL model to two TCTL models and then transfer these models to CTL. This step allows launching NuSMV checker as shown in Figure 7.19. After completing the transformation, we perform the third step, launching NuSMV checker to get the verification results. Figure 7.20 shows the screen of the verification results of the transformed model and the assigned formulae. In this screen, the tool provides for each CTL model time calculations of transforming the model, transforming the formulae, the verification process, and the total time.

After verifying the system using MV-Checker with launching NuSMV, we got the same verification results in Table 7.1. Table 7.11 presents the scalability results of the second experiment. This table shows the number of agents represented by (#Age.), the number of reachable states (#St.), the average time in milliseconds of transforming the two classical TCTL to CTL (Ave.Models (ms)), the average time of transforming the trust formula in each model (Ave.Formulae (ms)) and the average of the verification time. Although we got the same verification results in each experiment as we expected, Figure 7.21, which graphically represents the relation between the number of agents and the verification time for each experiment (Table 7.10 and Table 7.11) shows that transforming the 3v-TCTL to its classical case is more efficient and scalable than transforming it to CTL. The figure proves that verifying the transformed TCTL model using MCMAS^t took less time than verifying


CTL Model (Negative Cut)

```
( state = Pr7_10 & action = acc_c_7_10 & Fda_1-action = acc_c_7_10 & Manuf_1-action
( state = Pr7 & action = acc_7_11 & Fda_1-action = acc_7_11 & Manuf_1-action = acc_7
( state = Pr7_11 & action = acc_c_7_11 & Fda_1-action = acc_c_7_11 & Manuf_1-action
( state = Pr7 & action = acc_7_12 & Fda_1-action = acc_7_12 & Manuf_1-action = acc_7
( state = Pr7_12 & action = acc_c_7_12 & Fda_1-action = acc_c_7_12 & Manuf_1-action
( state = Pr8 & action = acc_8_9 & Fda_1-action = acc_8_9 & Manuf_1-action = acc_8_9
( state = Pr8_9 & action = acc_c_8_9 & Fda_1-action = acc_c_8_9 & Manuf_1-action = a
( state = Pr8 & action = acc_8_10 & Fda_1-action = acc_8_10 & Manuf_1-action = acc_8
( state = Pr8_10 & action = acc_c_8_10 & Fda_1-action = acc_c_8_10 & Manuf_1-action
( state = Pr8 & action = acc_8_11 & Fda_1-action = acc_8_11 & Manuf_1-action = acc_8
( state = Pr8_11 & action = acc_c_8_11 & Fda_1-action = acc_c_8_11 & Manuf_1-action
( state = Pr8 & action = acc_8_12 & Fda_1-action = acc_8_12 & Manuf_1-action = acc_8
( state = Pr8_12 & action = acc_c_8_12 & Fda_1-action = acc_c_8_12 & Manuf_1-action
( state = Pr9 & action = acc_9_10 & Fda_1-action = acc_9_10 & Manuf_1-action = acc_9
( state = Pr9_10 & action = acc_c_9_10 & Fda_1-action = acc_c_9_10 & Manuf_1-action
( state = Pr9 & action = acc_9_11 & Fda_1-action = acc_9_11 & Manuf_1-action = acc_9
( state = Pr9_11 & action = acc_c_9_11 & Fda_1-action = acc_c_9_11 & Manuf_1-action
( state = Pr9 & action = acc_9_12 & Fda_1-action = acc_9_12 & Manuf_1-action = acc_9
( state = Pr9_12 & action = acc_c_9_12 & Fda_1-action = acc_c_9_12 & Manuf_1-action
( state = Pr10 & action = acc_10_11 & Fda_1-action = acc_10_11 & Manuf_1-action = ac
( state = Pr10_11 & action = acc_c_10_11 & Fda_1-action = acc_c_10_11 & Manuf_1-ac
( state = Pr10 & action = acc_10_12 & Fda_1-action = acc_10_12 & Manuf_1-action = ac
( state = Pr10_12 & action = acc_c_10_12 & Fda_1-action = acc_c_10_12 & Manuf_1-ac
( state = Pr11 & action = acc_11_12 & Fda_1-action = acc_11_12 & Manuf_1-action = ac
( state = Pr11_12 & action = acc_c_11_12 & Fda_1-action = acc_c_11_12 & Manuf_1-ac

TRUE : state;
 esac)
```

Launch NuSMV
Verification Results
Time of Transofrming Model
Total Time of Transforming Formulas
Time of Verification Process
Total Time

CTL Model (Positive cut

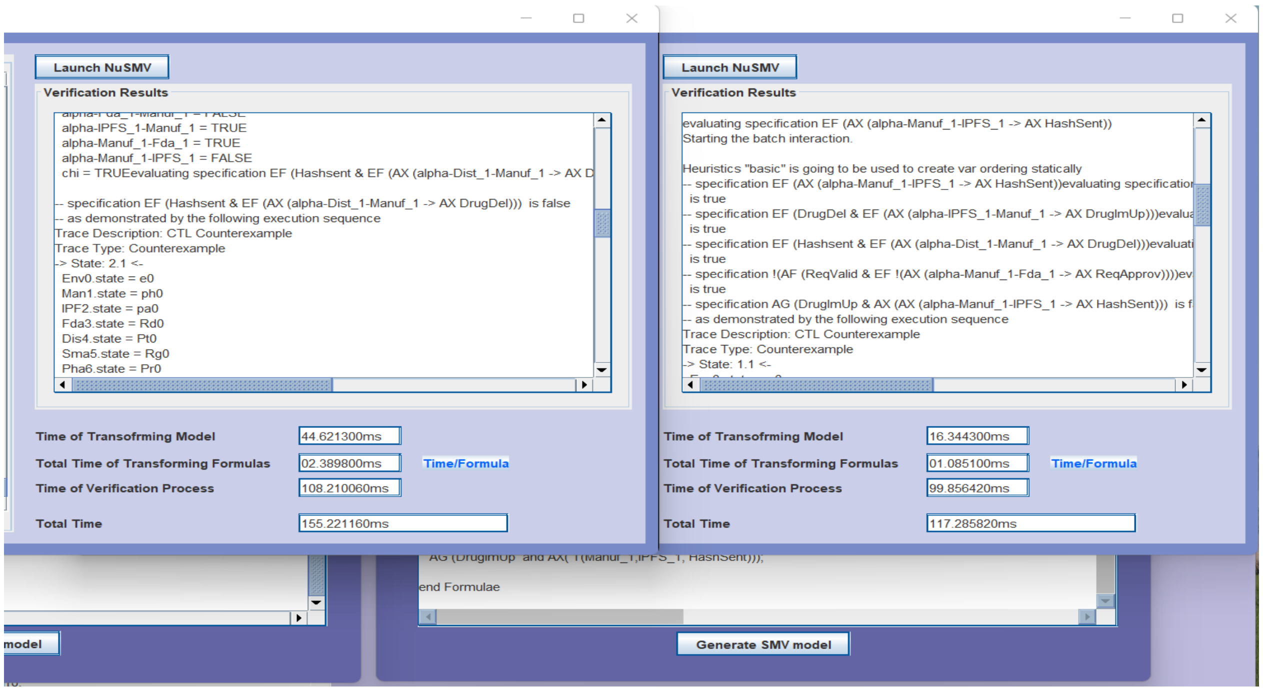
```
( state = Pr7_10 & action = acc_c_7_10 & Fda_1-action = acc_c_7_10 & Manuf_1-action
( state = Pr7 & action = acc_7_11 & Fda_1-action = acc_7_11 & Manuf_1-action = acc_7
( state = Pr7_11 & action = acc_c_7_11 & Fda_1-action = acc_c_7_11 & Manuf_1-action
( state = Pr7 & action = acc_7_12 & Fda_1-action = acc_7_12 & Manuf_1-action = acc_7
( state = Pr7_12 & action = acc_c_7_12 & Fda_1-action = acc_c_7_12 & Manuf_1-action
( state = Pr8 & action = acc_8_9 & Fda_1-action = acc_8_9 & Manuf_1-action = acc_8_9
( state = Pr8_9 & action = acc_c_8_9 & Fda_1-action = acc_c_8_9 & Manuf_1-action = a
( state = Pr8 & action = acc_8_10 & Fda_1-action = acc_8_10 & Manuf_1-action = acc_8
( state = Pr8_10 & action = acc_c_8_10 & Fda_1-action = acc_c_8_10 & Manuf_1-action
( state = Pr8 & action = acc_8_11 & Fda_1-action = acc_8_11 & Manuf_1-action = acc_8
( state = Pr8_11 & action = acc_c_8_11 & Fda_1-action = acc_c_8_11 & Manuf_1-action
( state = Pr8 & action = acc_8_12 & Fda_1-action = acc_8_12 & Manuf_1-action = acc_8
( state = Pr8_12 & action = acc_c_8_12 & Fda_1-action = acc_c_8_12 & Manuf_1-action
( state = Pr9 & action = acc_9_10 & Fda_1-action = acc_9_10 & Manuf_1-action = acc_9
( state = Pr9_10 & action = acc_c_9_10 & Fda_1-action = acc_c_9_10 & Manuf_1-action
( state = Pr9 & action = acc_9_11 & Fda_1-action = acc_9_11 & Manuf_1-action = acc_9
( state = Pr9_11 & action = acc_c_9_11 & Fda_1-action = acc_c_9_11 & Manuf_1-action
( state = Pr9 & action = acc_9_12 & Fda_1-action = acc_9_12 & Manuf_1-action = acc_9
( state = Pr9_12 & action = acc_c_9_12 & Fda_1-action = acc_c_9_12 & Manuf_1-action
( state = Pr10 & action = acc_10_11 & Fda_1-action = acc_10_11 & Manuf_1-action = ac
( state = Pr10_11 & action = acc_c_10_11 & Fda_1-action = acc_c_10_11 & Manuf_1-ac
( state = Pr10 & action = acc_10_12 & Fda_1-action = acc_10_12 & Manuf_1-action = ac
( state = Pr10_12 & action = acc_c_10_12 & Fda_1-action = acc_c_10_12 & Manuf_1-ac
( state = Pr11 & action = acc_11_12 & Fda_1-action = acc_11_12 & Manuf_1-action = ac
( state = Pr11_12 & action = acc_c_11_12 & Fda_1-action = acc_c_11_12 & Manuf_1-ac

TRUE : state;
 esac)
```

```
 AG (DrugImUp  and AX( T(Manuf_1,IPFS_1, HashSent)));

end Formulae
```

```
 AG (DrugImUp  and AX( T(Manuf_1,IPFS_1, HashSent)));

end Formulae
```

Figure 7.19: Screen shows the transforming of 3v-TCTL model with the formulae to TCTL and then to CTL to launch NuSMV

the transformed CTL using NuSMV. Moreover, the first is more scalable in increasing the number of agents, where we could reach 70 agents with a short increase in the verification time. On the other hand, with NuSMV, we reached six experiments with 56 agents and an exponential increase in the verification time.



```
alpha-IPFS_1-Manuf_1 = TRUE
alpha-Manuf_1-Fda_1 = TRUE
alpha-Manuf_1-IPFS_1 = FALSE
chi = TRUEevaluating specification EF (Hashsent & EF (AX (alpha-Dist_1-Manuf_1 -> AX D

-- specification EF (Hashsent & EF (AX (alpha-Dist_1-Manuf_1 -> AX DrugDel)))  is false
-- as demonstrated by the following execution sequence
Trace Description: CTL Counterexample
Trace Type: Counterexample
-> State: 2.1 <-
  Env0.state = e0
  Man1.state = ph0
  IPF2.state = pa0
  Fda3.state = Rd0
  Dis4.state = Pt0
  Sma5.state = Rg0
  Pha6.state = Pr0
```

Time of Transofrming Model: 44.621300ms
Total Time of Transforming Formulas: 02.389800ms
Time of Verification Process: 108.210060ms
Total Time: 155.221160ms

```
evaluating specification EF (AX (alpha-Manuf_1-IPFS_1 -> AX HashSent))
Starting the batch interaction.

Heuristics "basic" is going to be used to create var ordering statically
-- specification EF (AX (alpha-Manuf_1-IPFS_1 -> AX HashSent))evaluating specificatio
 is true
-- specification EF (DrugDel & EF (AX (alpha-IPFS_1-Manuf_1 -> AX DrugImUp)))evalu
 is true
-- specification EF (Hashsent & EF (AX (alpha-Dist_1-Manuf_1 -> AX DrugDel)))evaluati
 is true
-- specification !(AF (ReqValid & EF !(AX (alpha-Manuf_1-Fda_1 -> AX ReqApprov))))ev
 is true
-- specification AG (DrugImUp & AX (AX (alpha-Manuf_1-IPFS_1 -> AX HashSent)))  is f
-- as demonstrated by the following execution sequence
Trace Description: CTL Counterexample
Trace Type: Counterexample
-> State: 1.1 <-
```

Time of Transofrming Model: 16.344300ms
Total Time of Transforming Formulas: 01.085100ms
Time of Verification Process: 99.856420ms
Total Time: 117.285820ms

Figure 7.20: Verification results of verifying the transformed 3v-TCTL model to CTL using NuSMV

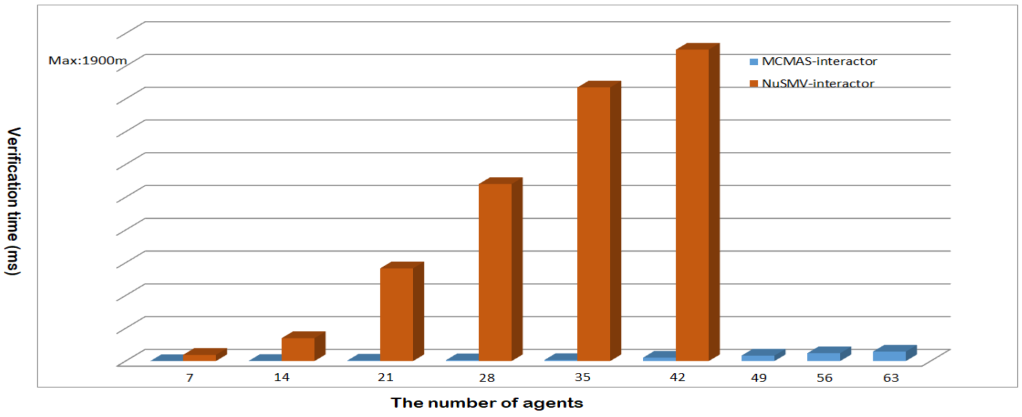


Figure 7.21: Comparison between the results of the 3v-TCTL verification approaches, the blue results using MCMAS^t and the brown results using NuSMV.

Table 7.11: Scalability results of running the MV-Checker six times over the two transformed CTL models

#Age.	**#St.**	**Ave.Models (ms)**	**Ave.Formulae (ms)**	**Ave.Verif(ms)**
7	43	43.41	02.67	37.93
14	1849	83.37	03.34	139.90
21	79,507	233.88	12.11	566.08
28	3,418,801	656.21	80.09	1080.61
42	1.47008e+06	1022.33	133.02	1670.11
56	6.32136e+18	1715.09	189.66	1900.89

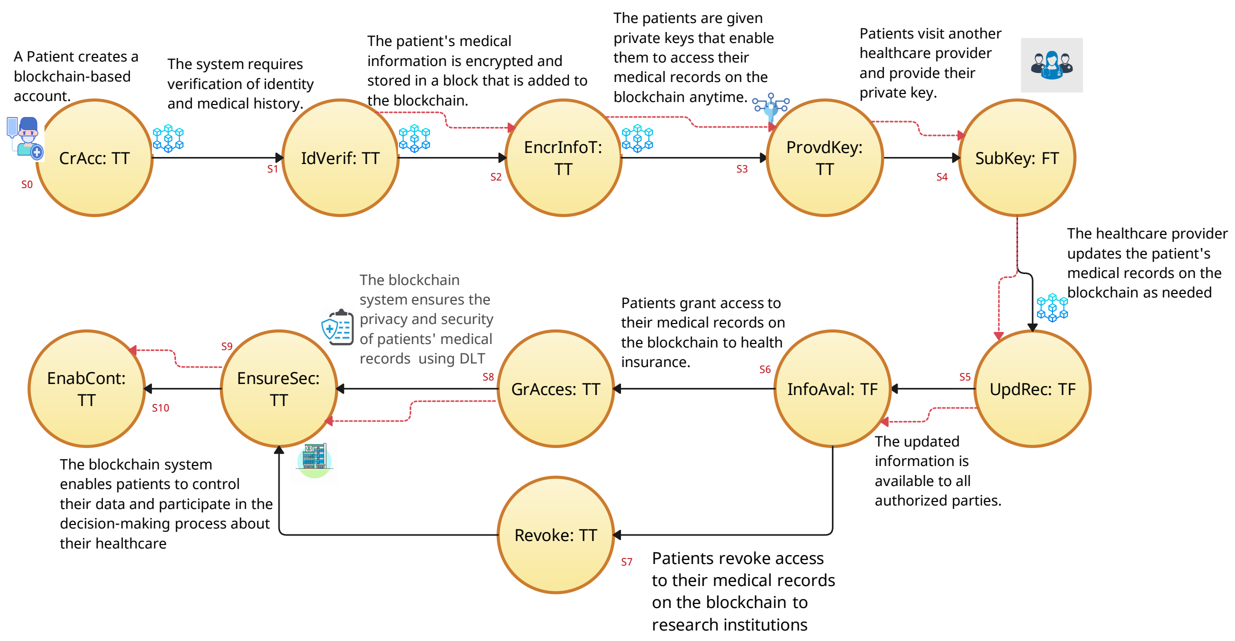
7.4 Modeling and Verifying Trust Systems with Inconsistency

7.4.1 Case Study 7: A 4v-TCTL Blockchain-Based Health Record System with Inconsistency

System Functionalities

We assume that the system is designed by two experts who have agreements and disagreements about the behaviours of the system. The designers agree (both say true) that the system starts at $S0$ when a patient creates a blockchain-based account. They also agree that in $S1$, the system requires identity verification from the patient and medical history. In $S2$ the designers agree that the patient's medical information is encrypted and stored in a block added to the blockchain. They also agree in $S3$ where the patents are given private keys, enabling them to access their medical records on the blockchain anytime. Then is $S4$, the designers disagree (the first says false and the second says true) that in case the patients visit another healthcare provider, they provide their private keys to the healthcare for authorized purposes. Next, in $S5$, the designers disagree (the first says true and the second says false) that the healthcare provider updates the patient's medical records on the blockchain as needed. The same in $S6$, the updated information becomes available to all authorized patients. Then in $S8$, the two designers agree that the patient granted healthcare providers access to their medical records on the blockchain. They also agree that in $S7$, patients can revoke access to their medical records on the blockchain. Then, in $S9$, they agree that the blockchain system ensures the privacy and security of the patient's medical records by using Distributed Ledger Technology (DLT). Finally, the designers agree that at $S10$, the system enables patients to control their data and participate in the decision-making process about their healthcare.

From the system model depicted in Figure 7.22, the trust relations between the system's agents are indicated in Red-dashed lines.



A Patient creates a blockchain-based account.
The system requires verification of identity and medical history.
The patient's medical information is encrypted and stored in a block that is added to the blockchain.
The patients are given private keys that enable them to access their medical records on the blockchain anytime.
Patients visit another healthcare provider and provide their private key.
CrAcc: TT
IdVerif: TT
EncrInfoT: TT
ProvdKey: TT
SubKey: FT
S0
S1
S2
S3
S4
The healthcare provider updates the patient's medical records on the blockchain as needed
The blockchain system ensures the privacy and security of patients' medical records using DLT
Patients grant access to their medical records on the blockchain to health insurance.
EnabCont: TT
EnsureSec: TT
GrAcces: TT
InfoAval: TF
UpdRec: TF
S9
S10
S8
S6
S5
The updated information is available to all authorized parties.
The blockchain system enables patients to control their data and participate in the decision-making process about their healthcare
Revoke: TT
S7
Patients revoke access to their medical records on the blockchain to research institutions

Figure 7.22: A Blockchain-Based Health Record system with 4v-TCTL under inconsistency

System Specifications

We verify seven trust properties, including safety, reachability and liveness properties, against the modelled system. The properties are interpreted in TCTL, and the atomic propositions and agents are expressed as follows: Agent "*Pat*" means "patient ", "*BloCh*" means “Blochchain ”, “*Hprov*” means “ health provider”, “*secHprov*” means “ second health provider visited by the patient”. The atomic proposition “*EncryInfo*” means “the blockchain encrypts the received information ”, “*ProvdKey*” means “the health provider provides to the patient the private key of the health record ”, “*SubKey*” means ”The patient shares their private key with the second health provider in order to grant access to their health records ”, “*UpdRec*” means “the health provider update the patient’s health record when needed ”, “*InfoAval*” means “ the blockchain makes the information available for access by the authorised parties”, “*Revoke*” means “the patient’s information is revoked ”, “*EnsureSec*” means “the blockchain secures the patient’s information ”, “*EnabCont*” means “ the blockchain enables patients to control their data and participate in the decision-making process about their healthcare ”

(1) *“ There is at least one pathway within the system where the patient trusts the blockchain to encrypt the received information. ”*

$\varphi_1 =$ *EF T((Pat1, BloCh1 ,EncryInfo)*

(2) *“There is at least one pathway within the system where the patient trusts the health provider to provide the private key of his/her record”.*

$\varphi_2 =$ *EF T((Pat1, Hprov1, ProvdKey)*

(3) *“In all possible executions, the second health provider trusts the patient to provide the private key to access the patient’s health record.”*

$\varphi_3 =$ *EF T((secHprov1, Pat1, SubKey)*

(4) *“There is at least one pathway within the system where the patient trusts the health provider to update her/her health record when needed”*

$\varphi_4 =$ *EF T((Pat1, Hprov1, UpdRec)*

(5) *“There is at least one pathway within the system where the health provider trusts the blockchain to make the information available for access”.*

$\varphi_5 =$ *EF T((Hprov1, BloCh1, InfoAval)*

(6) *"It is not the case that in all possible executions when the patient's information is revoked, the latter doesn't trust the blockchain to secure this information ".*

$\varphi_6 = \neg$ *AF (Revoke and EF !T(Pat1, BloCh1, EnsureSec))*

(7) *"There is at least one pathway within the system where the patient trusts the blockchain to enable information control by the patent".*

$\varphi_7 =$ *EF T(Pat1, BloCh1, EnabCont)*

System Verification

After modelling the system in Figure 7.22, we encoded this model into 4v-VISPLS, assigned a set of seven properties to be checked over this model, and saved the file. Then, we run MV-Checker to verify the system following similar steps and the same machine used in the first experiment. The difference here is that we reason about inconsistency over 4v- TCTL model encoded in 4v-ISPL. After verifying the two models using MCMAS^t, we determine the final results based on the following approximations: Assume that we save the verification result of each formula in the first classical model (considers TF is true) in x and the second model (considers FT is true) in y, then

- If $x = True$ and $y = True$ then, the final result: $TF \sqcup FT = TT$
- If $x = False$ and $y = True$ then, the final result: $\emptyset \sqcup FT = FT$
- If $x = True$ and $y = False$ then, the final result: $TF \sqcup \emptyset = TF$
- If $x = False$ and $y = False$ then, the final result: $\emptyset \sqcup \emptyset = FF$

The final results in Table 7.3 show that there is an agreement of (true, true) about the formulae φ_1, φ_3, φ_6 and φ_7. It also shows the disagreement about the formula φ_2 where the first designer said no and the other said yes (false, true). The opposite applies to the formulae φ_4 and φ_5.

Table 7.12: Verification results of the Blockchain-Based Health Record System with 4v-TCTL

Pro.	**TF**	**FT**	**Result**
φ_1	T	T	TT
φ_2	F	T	FT
φ_3	T	T	TT
φ_4	T	F	TF
φ_5	T	F	TF
φ_6	T	T	TT
φ_7	T	T	TT

We checked the scalability of the proposed approach for reasoning about inconsistency by performing seven experiments, running the MV-Checker on top of MCMAS^t. The results in Table 7.13 show the logarithmic increase in the verification time coinciding with adding five agents in each experiment. The average memory usage in megabytes for the two transformed classical models shows that the memory usage from the experiment with 20 agents and more increases exponentially.

Table 7.13: Scalability results after running the MV-Checker over the classical models seven times

#Age.	**#St**	**Ave.(MB)**	**Ave.time(ms)**
5	17	11.10288	0.02
10	95	13.1937	0.13
15	761	26.8433	0.62
20	6647	122.1288	3.66
25	59297	130.2536	10.55
30	1.77891e+02	142.2055	12.02
35	3.47991e+05	150.4207	20.45

7.5 Comparing MV-Checker Performance with the BT Tool for Verifying TCTL models

In this section, we compare the performance of MV-Checker with the BT tool introduced in [24]. This tool transforms the Branching Trust Logic (BT) into CTL to reuse the NuSMV checker. BT logic encompasses TCTL where the latter's modality $T(i, j, \phi)$ is a part of the firsts' modalities. Below, we illustrate the results of verifying the same case study used in the mentioned paper, which is a system of ordering protocol governing the interaction between seller and buyer agents. The results obtained using MV-Checker are compared with the ones obtained using the BT tool on the same machine with the following specifications: Intel(R) Core(TM) i7-6500U CPU with 2.50GHz, 2.59 GHz Installed RAM 16.0 GB. Figure 7.23 presents the increase in the total time of transforming the model and formula and the verification time regarding the increase in the number of agents. The experiments started with 3 and ended with 15 agents. In each experiment, three agents are added regarding the system and formula. The figure shows that the total time used in MV-Checker (indicated in blue) is less than in the BT tool (indicated in brown). Although the difference in verification time between the two tools is not substantial, MV-Checker demonstrated its proficiency in handling various types of logic, as detailed in Table 6.1. Furthermore, the checker boasts user-friendly interfaces, supplemented with a user-guide, further enhancing its usability.

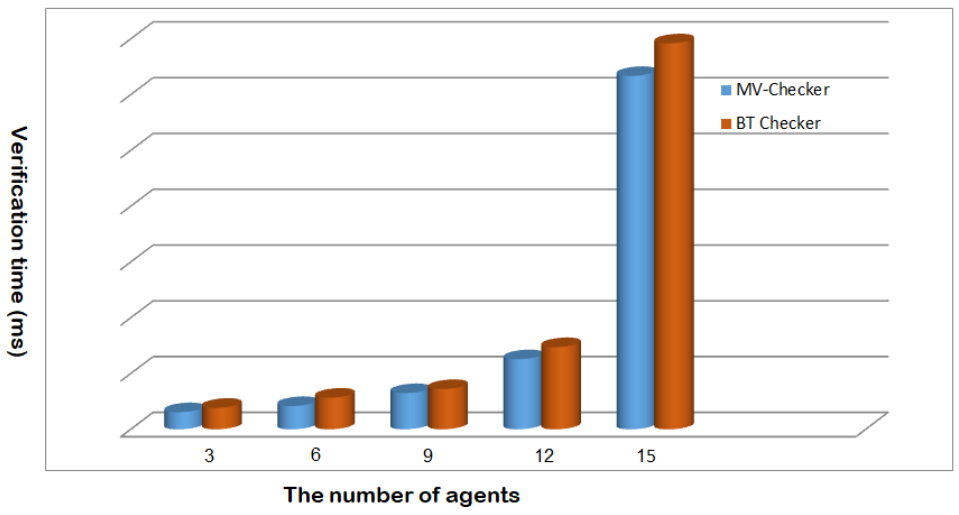


Figure 7.23: Comparing the performance of MV-Checker with the BT Java toolkit

Chapter 8

Conclusion and Future Work

8.1 Conclusion

Intelligent systems (IS) and IoTs are highly susceptible to uncertainty and inconsistency due to the intense and intricate interactions among their autonomous components (or agents). This makes their verification theoretically and practically challenging, especially when these applications involve essential protocols such as trust and commitment. The primary contributions combined with this research are addressing the following problems: (1) defining a computationally grounded semantics for agent communication regarding social commitments and trust under uncertainty and inconsistency; (2) producing multi-valued model checking MAS techniques with focus on IoT and IS systems with commitment-based and trust-based protocols; (3) developing an efficient model checking tool that implements these techniques.

In summary, building upon the technical background introduced in Chapter 2, our initial step was to delve into the cutting-edge research regarding the application of computational logics in formulating formal semantics for agent communication, leveraging social commitments and trust and their associated actions. We focused on the recently introduced logics of commitment CTLC and trust TCTL. We conducted an in-depth examination of methodologies addressing the challenges of verifying Multi-Agent Systems (MAS) amidst inconsistency and uncertainty, employing multi-valued logic mv-CTL. We systematically evaluated various prominent proposals against eight crucial criteria, highlighting their strengths and limitations.

Subsequently, in Chapter 3, we provided a new practical experiment of applying mv-CTL to modelling and verifying the IoT systems with missing information. From this experiment, we found that the existing approach has high efficiency in verifying open IoT systems. However, this approach deals only with systems that do not involve commitment or trust protocols. Based on these limitations, in Chapter 4, we started to produce new frameworks that integrate the mv-CTL with CTLC logic of commitment and introduced frameworks for Modeling and Verifying IoT and IS systems with mv-Commitment Logics. Specifically, we presented our new multi-valued logics of commitment: (1) 3v-CTL^{cc} and (2) 4v-CTL^{cc} for reasoning about uncertainty and inconsistency, respectively, over MAS, focusing on IoT/IS, with conditional commitment protocols. (3) 3v-CTL^{c} and (4) 4v-CTL^{c} for reasoning about uncertainty and inconsistency, respectively, over the same systems with unconditional commitment protocols. Using the new logic, we modelled several IoT and IS systems and assigned sets of commitment specifications to be checked against these systems. Moreover, we introduced a formal input language of the multi-valued models named mv-ISPL+. Then, because of the high complexity of the direct multi-valued model checking algorithms and to take advantage of the reduction ones by reusing the existing efficient model checking tools NuSMV and MCMAS+, we developed new reduction algorithms. These algorithms reduce the multi-valued logic of commitment to its classical case (CTLC) and to CTL. Moreover, we developed a new Java toolkit that implements these algorithms and automatically interacts with NuSMV and MCAMS+. Furthermore, we provided soundness proofs for our algorithms with comprehensive computational complexity analysis.

Sequentially, in Chapter 5, we presented our new multi-valued logics of trust: (5) 3v-TCTL and (6) 4v-TCTL for reasoning about uncertainty and inconsistency, respectively, over the systems under consideration with trust protocols. Following the same strategies with the multi-valued commitment logic, we modelled several IoT and IS systems with trust using our new logics. We introduced a new formal input language of the multi-valued trust models named mv-VISPL. Moreover, we developed reduction algorithms to model-check our systems against sets of trust specifications. These algorithms transform mv-TCTL into TCTL and CTL and facilitate using NuSMV and MCMAS^t checkers. We also developed a new tool to implement these algorithms and provided sound proofs and comprehensive computational complexity analysis.

Our findings proved that all the presented algorithms are sound and the approaches of the multi-valued model checking mv-CTLC and mv-TCTL are efficient as their time complexity is P-complete and their space complexity is PSPACE-complete, which means polynomial in both time and space.

Focusing on the implementation part, in Chapter 6, we enhanced the implementation by developing a new general tool named MV-Checker, which effectively verifies both multi-valued trust and commitment models designed under uncertain or inconsistent sources of knowledge. We provided a detailed explanation of the internal design and the functionalities of this tool with verification examples. Moreover, We provided the packages of all our case studies containing 12 experiments with SMV, ISPL+, VISPL, mv-ISPL+ and mv-VISPL files. Additionally, we provided the source code, the Jar file of the tool, and the user manual document explaining the use of the proposed tool.

To evaluate the practicality, scalability and efficiency of our approaches and the performance of our new checker, in Chapter 7, we applied them to several case studies that cover modelling and verifying open and complex systems with uncertainty and inconsistency settings in IoT and IS domains. The reliable and accurate results of all the experiments performed in this research proved the scalability and the high practicality and efficiency of our approaches. Moreover, we compared our results using the reduction algorithms from the mv-CTLC and mv-TCTL to CTL and their classical cases. Based on these comparisons, we found that, even though the two main reduction approaches gave accurate and reliable results, reduction to the classical versions and using MCMAS is more scalable and takes less verification time than reduction to CTL and using NuSMV checker. We also found that this shortcoming resulted from the freshly added states and atomic propositions in the transformation process explained in Algorithm 4. Based on this, we set plans for NuSMV improvement in future work that will be explained in the following section.

For evaluating the performance of the developed MV-Checker, due to the lack of existing model checkers that deal with multi-valued logics, we compared the performance of the checker with another tool named BT checker by verifying a TCTL model using a reduction algorithm over both tools. The results showed that the total time used in the verification process over MV-Checker is less than with BT. Although the difference in the total time was not initial, MV-Checker is still more efficient than the BT tool in terms of the number of functions that can handle and the flexibility in dealing with different logics, including the classical and the multi-valued ones.

8.2 Future Work

Finally, in this chapter, we set future work for extending our performed work and some open issues that are not considered in this thesis:

Work extension:

- As we addressed verifying an open MAS with multiple data sources with the presence of uncertainty or inconsistency separately, we plan to produce new logics with new semantics to capture the presence of both uncertainty and inconsistency in the same system. This logic depends on the underlying 9-valued lattice ($L_{3\times3}$). Analogous to the 2x2 and 3v-logics we used for reasoning about inconsistency and uncertainty, this nine-valued logic is derived from the product algebra 3x3. It can be used for reasoning about disagreement between two sources and also allows missing information (uncertainty) in each source within the same system.
- We plan to add more functionalities to the MV-Checker tool:
  - verifying systems modelled with nine-valued logic.
  - verifying systems modelled with the combined trust and commitment TCTLC logic [9].

  - transforming multi-valued models with multiple truth values over transitions between states, not only in states.
  - Adopting AI technology to facilitate the verification process by automatically generating ISPL and mv-ISPL files from a given FSM. The general steps for this technique are:
    1. Image Recognition: The first step is to use an image recognition or computer vision model to identify and extract the relevant information from the image. In this case, the goal would be to recognize the FSM structure, states, transitions, and connections.
    2. Structure Extraction: Once the FSM structure is recognized, the AI model needs to extract information about states, transitions, and their relationships.
    3. Model Representation: The extracted information then needs to be converted into a format that can be used for model checking, such as ISPL. This involves creating a text file that accurately represents the FSM.
    4. Validation and Correction: It's important to note that the AI-generated representation may not always be perfect. There may be errors or ambiguities in the extracted information. Therefore, a validation step may be necessary to ensure the accuracy of the generated ISPL code. This could involve human review or additional automated checks.
    5. Model Checker Input: Finally, the generated ISPL code can be input into a model checker for verification and analysis.

- We plan to develop a new version of NuSMV to deal directly with trust and commitment syntax and semantics.

Open issues

- We plan to conduct a deep investigation from both conceptual and implementation perspectives to take a new direction and extend the Alternating-time temporal logic (ATL) with trust and commitment modalities considering the multi-valued settings.

- We also plan to conduct a deep investigation in producing a new multi-valued Logic of Allies and Enemies [47]. This logic outlines the behavior of relationships within a social network, assuming that agents tend to form more enduring connections with others in the network. Producing the multi-valued version of this logic opens a wide door for reasoning about social network graphs with missing and inconsistent information.

Bibliography

[1] Aghdam, Z.N., Rahmani, A.M., Hosseinzadeh, M.: The role of the internet of things in healthcare: Future trends and challenges. Computer methods and programs in biomedicine **199**, 105903 (2021)

[2] Alwhishi, G., Bentahar, J., Drawel, N.: Reasoning about uncertainty over IoT systems. In: 2022 International Wireless Communications and Mobile Computing (IWCMC), pp. 306–311. IEEE (2022)

[3] Alwhishi, G., Bentahar, J., Elwhishi, A.: Verifying timed commitment specifications for IoT-cloud systems with uncertainty. In: 2022 9th International Conference on Future Internet of Things and Cloud (FiCloud), pp. 173–180. IEEE (2022)

[4] Alwhishi, G., Bentahar, J., Elwhishi, A.: Multi-valued model checking a smart glucose monitoring system with trust. In: 2023 International Wireless Communications and Mobile Computing (IWCMC), pp. 1697–1702. IEEE (2023)

[5] Alwhishi, G., Bentahar, J., Elwhishi, A.: Three-valued model checking smart contract systems with trust under uncertainty. In: The International Conference on Deep Learning, Big Data and Blockchain, pp. 119–133. Springer (2023)

[6] Alwhishi, G., Bentahar, J., Elwhishi, A., Pedrycz, W.: Mv-checker: A software tool for multi-valued model checking intelligent applications with trust and commitment. Available at SSRN 4505659 (2023)

[7] Alwhishi, G., Bentahar, J., Elwhishi, A., Pedrycz, W., Drawel, N.: Multi-valued model checking iot and intelligent systems with commitment protocols in multi-source data environments. Information Fusion p. 102048 (2023)

[8] Alwhishi, G., Drawel, N., Bentahar, J.: Model checking intelligent information systems with 3-valued timed commitments. In: International Conference on Mobile Web and Intelligent Information Systems, pp. 237–251. Springer (2022)

[9] Baharloo, N., Bentahar, J., Alwhishi, G., Drawel, N., Pedrycz, W.: Verifying trust over iot-ad hoc network-based applications under uncertainty. Available at SSRN 4524628

[10] Baier, C., Katoen, J.: Principles of model checking. MIT Press (2008)

[11] Belnap Jr, N.D.: A useful four-valued logic. In: Modern uses of multiple-valued logic, pp. 5–37. Springer (1977)

[12] Bidgoly, A.J., Ladani, B.T.: Trust modeling and verification using colored petri nets. In: 2011 8th International ISC Conference on Information Security and Cryptology, pp. 1–8. IEEE (2011)

[13] Bruns, G., Godefroid, P.: Model checking partial state spaces with 3-valued temporal logics. In: International conference on computer aided verification, pp. 274–287. Springer (1999)

[14] Bruns, G., Godefroid, P.: Generalized model checking: Reasoning about partial state spaces. In: International Conference on Concurrency Theory, pp. 168–182. Springer (2000)

[15] Bruns, G., Godefroid, P.: Model checking with multi-valued logics. In: International Colloquium on Automata, Languages, and Programming, pp. 281–293. Springer (2004)

[16] Carter, S.M., Rogers, W., Win, K.T., Frazer, H., Richards, B., Houssami, N.: The ethical, legal and social implications of using artificial intelligence systems in breast cancer care. The Breast **49**, 25–32 (2020)

[17] Chechik, M., Devereux, B., Easterbrook, S., Gurfinkel, A.: Multi-valued symbolic model-checking. ACM Transactions on Software Engineering and Methodology (TOSEM) **12**(4), 371–408 (2003)

[18] Chechik, M., Easterbrook, S., Petrovykh, V.: Model-checking over multi-valued logics. In: FME 2001: Formal Methods for Increasing Software Productivity: International Symposium of Formal Methods Europe Berlin, Germany, March 12–16, 2001 Proceedings, pp. 72–98. Springer (2001)

[19] Davey, B.A., Priestley, H.A.: Introduction to lattices and order. Cambridge university press (2002)

[20] Desai, N., Cheng, Z., Chopra, A.K., Singh, M.P.: Toward verification of commitment protocols and their compositions. In: Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems, pp. 1–3 (2007)

[21] Drawel, N.: Model checking trust-based multi-agent systems. Ph.D. thesis, Concordia University (2019)

[22] Drawel, N., Bentahar, J., El-Menshawy, M., Laarej, A.: Verifying temporal trust logic using ctl model checking. In: TRUST@ AAMAS, pp. 62–74 (2018)

[23] Drawel, N., Bentahar, J., Laarej, A., Rjoub, G.: Formalizing group and propagated trust in multi-agent systems. In: Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI, pp. 60–66 (2020)

[24] Drawel, N., Bentahar, J., Laarej, A., Rjoub, G.: Formal verification of group and propagated trust in multi-agent systems. Autonomous Agents and Multi-Agent Systems **36**(1), 1–31 (2022)

[25] Drawel, N., Bentahar, J., Laarej, A., Rjoub, G.: Formal verification of group and propagated trust in multi-agent systems. Autonumos Agents Multi-Agent Systmes **36**(1), 19 (2022). DOI 10.1007/s10458-021-09542-6. URL `https://doi.org/10.1007/s10458-021-09542-6`

[26] Drawel, N., Bentahar, J., Qu, H.: Degrees of trust: Temporal logic and model checking. TRUST@ AAMAS pp. 62–74 (2019)

[27] Drawel, N., Bentahar, J., Shakshuki, E.: Reasoning about trust and time in a system of agents. Procedia Computer Science **109**, 632–639 (2017)

[28] Drawel, N., Laarej, A., Bentahar, J., El Menshawy, M.: Transformation-based model checking temporal trust in multi-agent systems. Journal of Systems and Software p. 111383 (2022)

[29] Drawel, N., Qu, H., Bentahar, J., Shakshuki, E.: Specification and automatic verification of trust-based multi-agent systems. Future Generation Computer Systems **107**, 1047–1060 (2020)

[30] Easterbrook, S., Chechik, M.: A framework for multi-valued reasoning over inconsistent viewpoints. In: Proceedings of the 23rd International Conference on Software Engineering. ICSE 2001, vol. 1, pp. 411–420. Citeseer (2001)

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

[32] El Kholy, W., Bentahar, J., El Menshawy, M., Qu, H., Dssouli, R.: Modeling and verifying choreographed multi-agent-based web service compositions regulated by commitment protocols. Expert systems with applications **41**(16), 7478–7494 (2014)

[33] El Kholy, W., El Menshawy, M., Bentahar, J., Qu, H., Dssouli, R.: Representing and reasoning about communicative conditional commitments. In: Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems, pp. 1169–1170 (2013)

[34] El Kholy, W., El Menshawy, M., Bentahar, J., Qu, H., Dssouli, R.: Verifying multiagent-based web service compositions regulated by commitment protocols. In: 2014 IEEE International Conference on Web Services, pp. 49–56. IEEE (2014)

[35] El-Menshawy, M., Bentahar, J., Dssouli, R.: Verifiable semantic model for agent interactions using social commitments. In: International Workshop on Languages, Methodologies and Development Tools for Multi-Agent Systems, pp. 128–152. Springer (2009)

[36] El-Menshawy, M., Bentahar, J., Dssouli, R.: Symbolic model checking commitment protocols using reduction. In: International Workshop on Declarative Agent Languages and Technologies, pp. 185–203. Springer (2010)

[37] El-Menshawy, M., Bentahar, J., Dssouli, R.: Model checking commitment protocols. In: International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems, pp. 37–47. Springer (2011)

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

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

[40] El-Menshawy, M., Bentahar, J., El Kholy, W., Laarej, A.: Model checking real-time conditional commitment logic using transformation. Journal of Systems and Software **138**, 189–205 (2018)

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

[42] El-Qurna, J., Yahyaoui, H., Almulla, M.: A new framework for the verification of service trust behaviors. Knowledge-Based Systems **121**, 7–22 (2017)

[43] Emerson, E.A.: Temporal and modal logic. In: J. van Leeuwen (ed.) Handbook of Theoretical Computer Science, vol. B., pp. 955–1072. MIT Press (1990)

[44] Fainekos, G.E.: An introduction to multi-valued model checking. Tech. rep., University of Pennsylvania, Department of Computer and Information Science, Technical Report No. MS-CIS-05-16 (2005)

[45] Fang, Z., Fu, H., Gu, T., Qian, Z., Jaeger, T., Hu, P., Mohapatra, P.: A model checking-based security analysis framework for IoT systems. High-Confidence Computing **1**(1), 100004 (2021)

[46] Gurfinkel, A., Chechik, M.: Multi-valued model checking via classical model checking. In: International conference on concurrency theory, pp. 266–280. Springer (2003)

[47] Van der Hoek, W., Kuijer, L., Wáng, Y.: Logics of allies and enemies: A formal approach to the dynamics of social balance theory. In: proceedings of the twenty-ninth international joint conference on artificial intelligence, vol. 2021, pp. 210–216 (2020)

[48] Hopgood, A.A.: Intelligent systems for engineers and scientists: a practical guide to artificial intelligence. CRC press (2021)

[49] Jamroga, W., Konikowska, B., Kurpiewski, D., Penczek, W.: Multi-valued verification of strategic ability. Fundamenta Informaticae **175**(1-4), 207–251 (2020)

[50] Jamroga, W., Konikowska, B., Penczek, W.: Multi-valued verification of strategic ability. In: Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems, pp. 1180–1189 (2016)

[51] Kleene, S.C.: Introduction to metamathematics (1952)

[52] Konikowska, B., Penczek, W.: Reducing model checking from multi-valued ctl* to ctl. In: International Conference on Concurrency Theory, pp. 226–239. Springer (2002)

[53] Konikowska, B., Penczek, W.: Model checking for multi-valued computation tree logics. In: Beyond two: theory and applications of multiple-valued logic, pp. 193–210. Springer (2003)

[54] Kozen, D.: Results on the propositional μ-calculus. Theoretical computer science **27**(3), 333–354 (1983)

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

[56] Laghari, A.A., Wu, K., Laghari, R.A., Ali, M., Khan, A.A.: A review and state of art of internet of things (iot). Archives of Computational Methods in Engineering pp. 1–19 (2021)

[57] Li, Y., Droste, M., Lei, L.: Model checking of linear-time properties in multi-valued systems. Information Sciences **377**, 51–74 (2017)

[58] Li, Y., Lei, L., Li, S.: Computation tree logic model checking based on multi-valued possibility measures. Information Sciences **485**, 87–113 (2019)

[59] Li, Y., Yang, G., Su, Z., Li, S., Wang, Y.: Human activity recognition based on multienvironment sensor data. Information Fusion **91**, 47–63 (2023)

[60] McKenzie, R.N., McNulty, G.F., Taylor, W.F.: Algebras, lattices, varieties, vol. 383. American Mathematical Soc. (2018)

[61] Michael, W.: An introduction to multiagent systems (2002)

[62] Mohamed, E.: The relation of artificial intelligence with internet of things: A survey. Journal of Cybersecurity and Information Management **1**(1), 30–24 (2020)

[63] Mostafa, N., Hamdy, W., Alawady, H.: Impacts of internet of things on supply chains: a framework for warehousing. Social sciences **8**(3), 84 (2019)

[64] Musamih, A., Salah, K., Jayaraman, R., Arshad, J., Debe, M., Al-Hammadi, Y., Ellahham, S.: A blockchain-based approach for drug traceability in healthcare supply chain. IEEE access **9**, 9728–9743 (2021)

[65] Nguyen, D.C., Ding, M., Pathirana, P.N., Seneviratne, A., Li, J., Niyato, D., Dobre, O., Poor, H.V.: 6g internet of things: A comprehensive survey. IEEE Internet of Things Journal **9**(1), 359–383 (2021)

[66] Pan, H., Li, Y., Cao, Y., Ma, Z.: Model checking computation tree logic over finite lattices. Theoretical computer science **612**, 45–62 (2016)

[67] Peled, E.M.C.O.G.D.A.: Model Checking. Cyber Physical Systems Series. MIT Press (1999)

[68] Ratta, P., Kaur, A., Sharma, S., Shabaz, M., Dhiman, G.: Application of blockchain and internet of things in healthcare and medical sector: applications, challenges, and future perspectives. Journal of Food Quality **2021**, 1–20 (2021)

[69] Rejeb, A., Simske, S., Rejeb, K., Treiblmaier, H., Zailani, S.: Internet of things research in supply chain management and logistics: A bibliometric analysis. Internet of Things **12**, 100318 (2020)

[70] Riedl, M.O.: Human-centered artificial intelligence and machine learning. Human behavior and emerging technologies **1**(1), 33–36 (2019)

[71] Roman, S.: Lattices and ordered sets. Springer Science & Business Media (2008)

[72] Rose, K., Eldridge, S., Chapin, L.: The internet of things: An overview. The internet society (ISOC) **80**, 1–50 (2015)

[73] Rosenmann, A.: A multiple-valued logic approach to the design and verification of hardware circuits. Journal of applied logic **15**, 69–93 (2016)

[74] Sami, H., Otrok, H., Bentahar, J., Mourad, A.: AI-based resource provisioning of IoE services in 6G: A deep reinforcement learning approach. IEEE Transactions on Network and Service Management **18**(3), 3527–3540 (2021)

[75] Savaglio, C., Ganzha, M., Paprzycki, M., Bădică, C., Ivanović, M., Fortino, G.: Agent-based internet of things: State-of-the-art and research challenges. Future Generation Computer Systems **102**, 1038–1053 (2020)

[76] Shoham, S., Grumberg, O.: Multi-valued model checking games. In: International Symposium on Automated Technology for Verification and Analysis, pp. 354–369. Springer (2005)

[77] Shoham, S., Grumberg, O.: Multi-valued model checking games. Journal of Computer and System Sciences **78**(2), 414–429 (2012)

[78] Timm, N., Gruner, S.: Parameterised three-valued model checking. Science of Computer Programming **126**, 94–110 (2016)

[79] Tissaoui, A., Saidi, M.: Uncertainty in IoT for smart healthcare: Challenges and opportunities. In: International Conference on Smart Homes and Health Telematics, pp. 232–239. Springer (2020)

[80] Wang, J., Chen, M., Zhou, J., Li, P.: Data communication mechanism for greenhouse environment monitoring and control: An agent-based iot system. Information Processing in Agriculture **7**(3), 444–455 (2020)

[81] Wang, X., Liu, J., Moore, S.J., Nugent, C.D., Xu, Y.: A behavioural hierarchical analysis framework in a smart home: Integrating HMM and probabilistic model checking. Information Fusion (2023). DOI https://doi.org/10.1016/j.inffus.2023.02.025

[82] Wang, Y., Singh, M.P.: Formal trust model for multiagent systems. In: IJCAI, vol. 7, pp. 1551–1556 (2007)

[83] Wooldridge, M., Jennings, N.R.: Intelligent agents: Theory and practice. The knowledge engineering review **10**(2), 115–152 (1995)

[84] Xu, Y., Ruan, D., Qin, K., Liu, J.: Lattice-valued logic. Studies in fuzziness and soft computing **132** (2003)