Formal Methods Enhanced Deep Learning for Integrated Cyber-Physical Systems

---

A

Dissertation

Presented to

the faculty of the School of Engineering and Applied Science

University of Virginia

---

in partial fulfillment

of the requirements for the degree

Doctor of Philosophy

by

Meiyi Ma

August 2021

APPROVAL SHEET

This

Dissertation

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

Doctor of Philosophy

Author: Meiyi Ma

This Dissertation has been read and approved by the examing committee:

Advisor: John Stankovic

Advisor: Lu Feng

Committee Member: Haiying Shen

Committee Member: Aidong Zhang

Committee Member: John Lach

Committee Member:

Committee Member:

Committee Member:

Accepted for the School of Engineering and Applied Science:

Jennifer L. West, School of Engineering and Applied Science

August 2021

Abstract

AI-powered Cyber-Physical Systems (CPS) form the basis of emerging and future smart services, improve quality of life, and bring advances in many critical areas, including intelligent transportation, robotics, smart cities and smart healthcare. The development of faster and more reliable networks accelerates the integration of Cyber-Physical Systems (i-CPS). A central problem is how to build reliable i-CPS facing significant new challenges due to the increasing integration, complexity, and environmental uncertainties. Targeting these challenges, in this dissertation, we develop rigorous and robust models for reliable i-CPS by integrating formal methods and deep learning with the following main contributions.

First, we conduct large-scale data-driven analytic for i-CPS, including the first study on over 1000 city requirements and study on uncertainty in i-CPS. Secondly, we develop the first decision support system for conflict detection and resolution for smart cities, which significantly improves cities' safety and performance in our experiments and simulation. Thirdly, we create novel formal specification languages and efficient runtime verification techniques for large-scale i-CPS with theoretical guarantees. The new specification languages achieve over 95% expressiveness coverage on real-world city requirements, while state-of-the-art only have 43% coverage. Additionally, we build a runtime monitoring tool to support specifying and monitoring requirements by different stakeholders in i-CPS. Fourthly, we develop novel formal methods enhanced deep learning techniques to increase the robustness of sequential prediction by incorporating formal specification and verification into the learning process. The framework is general to be applied to various deep sequential prediction models for i-CPS and significantly outperforms state-of-the-art. At last, we develop the first approach for calibrating the uncertainty estimation in Bayesian Recurrent Neural Networks (RNNs) through predictive monitoring with critical system requirements. It achieves a much higher quality of uncertainty estimation comparing to the baseline approaches.

To my mom.

Acknowledgments

I had an incredible Ph.D. journey with the kind support and help of many people. When I first started this journey, all I had was just a dream. I want to extend my sincere appreciation to all of them for making my dream come true.

First and foremost, I would like to express my deepest gratitude to my advisors, Dr. Jack Stankovic and Dr. Lu Feng, for their constant guidance, inspiration, encouragement, and support. I feel fortunate to have two exceptional advisors. Jack led me into academia. He taught me how to do research and trained me to become an independent researcher. He always inspired me, encouraged me, and believed in me even when I was not confident. I would never have become a Ph.D. and a faculty member without Jack's constant support and encouragement. Lu has been a role model to me for her passion and dedication to research. I am grateful for her constant support from research to life. One of the most cherished memories in my Ph.D. journey was when Lu and I worked on research papers together day and night. She was meticulous in every technical detail and every single word in writing, which deeply inspired and influenced me.

I was very fortunate to receive great support from my Ph.D. committee members and mentors in the research community. First, I would like to thank my committee members, Dr. Haiying Shen, Dr. Aidong Zhang, Dr. John Lach, and Dr. Alf Weaver, for their support of my dissertation and Ph.D. study. I sincerely appreciate their insightful questions, valuable suggestions, and precious time. Furthermore, I would like to thank Dr. John Knight, Dr. Hongning Wang, Dr. Chenyang Lu, and Dr. Hengchang Liu for their guidance, advice, and support for my research and career development at the different stages of my Ph.D. journey.

My thanks also go to many outstanding collaborators. I want to express my special thanks to Ji Gao, my husband and collaborator, for his company, support, and love. Ji and I started our Ph.D.

journey together. We often read papers, discussed research ideas, and worked overnight together. We learned so much from each other. Life is full of ups and downs, but he always lightens up my day. Next, I would like to thank Dr. Sarah Preum. Ever since the first day I joined Jack's group, Sarah has been there for me and treated me like a sister. She always generously shared her experience with me and encouraged me in each step of my Ph.D. journey. Furthermore, I was also fortunate to have collaborated with Dr. Ezio Bartocci on several papers. His knowledge and rigorous attitude in research deeply affected me. At last, this dissertation also benefited greatly from my research collaborations with Dr. Shan Lin, Dr. Desheng Zhang, Dr. Miao Fei, Dr. Donna Spruijt-Metz, Dr. Kayla de la Haye, Dr. Brooke Bell, Dr. Ifat Emi, Dr. Abu Sayeed, Dr. Asif Salekin, Dr. Mohsin Ahmed, Eli Lifland, Yukun Yuan, Wenqiang Chen, and Ye Gao.

Last but not least, I would like to thank my mom for her unconditional love and endless supports. She has made a lot of efforts and sacrifices to raise me. Despite her deepest desire to keep me, her only daughter around her, she still always supports me in pursuing my dreams. Everything I achieved today, I owe to her.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

As Artificial Intelligence (AI) technologies advance, deep learning has been broadly applied to Cyber-Physical Systems (CPS) and the Internet of Things (IoT), which form the basis of emerging and future services, improve quality of life, and bring advances in personalized health care, robotics, and smart cities. It is estimated by Statista that there will be 75 billion IoT-connected devices by 2025 [1]. The development of faster and more reliable networks, especially with the extensive advancing of 5G [2], accelerates the number of Integrated Cyber-Physical Systems (i-CPS).

While significant research efforts have been spent towards building smarter services, sensors, and infrastructures in CPS (e.g., smart cities), the research challenge of how to ensure that its real-time operations satisfy safety and performance requirements has only received scant attention. Most of the services are developed independently by different stakeholders under their own context. Integrating multiple CPS into the same environment (i.e., i-CPS) could cause problems that are not for-seen at their design time, such as, conflicts among their actions. For instance, a congestion control service may change a traffic light system to improve traffic due to a football game; whereas an environmental control service may change the same traffic light system due to noise and pollution emission. These two actions may lead to (1) a direct conflict between these two services or with a third service, e.g., an ambulance service is delayed due to conflicting traffic light schedules; (2) an environmental conflict with another service, e.g., an pollution emission control service of a power plant that was not aware of the increased pollution emission due to unexpected increase in traffic and did not update their emission setting, which lead to the combined emission surpassing the safety requirement. Such

conflicts could lead to severe consequences and affect millions of citizens lives every day, e.g., delayed travel time, exceeded air pollution levels, and pedestrian safety.

Moreover, a majority of services are supported by deep learning models. However, current deep learning techniques are not mature enough to deal with the challenges of i-CPS conflicts: safety-critical, large-scale, highly uncertain, and with humans in the loop. One of the biggest reasons is that, as data-driven approaches, they often do not consider system properties, environment requirements, or the complex interaction with other CPS or humans under the same environment. Their reliability then becomes questionable when being deployed in real-world environments.

On the other hand, as mathematically rigorous techniques, formal methods have been widely applied to verifying and evaluating critical systems such as aircraft. This motivates a novel research direction of verifying machine learning using formal methods in recent years, mainly targeting applications with well-specified safety requirements, such as autonomous driving and robotics. However, verification is only meaningful when paired with high-quality formal specification. The problem with current approaches is that such specification does not exist or is highly under-explored in i-CPS, especially in smart cities and health care domains. It creates great challenges to apply formal methods to machine learning towards i-CPS.

Research Questions: How to build reliable AI-powered i-CPS becomes a key research question for this dissertation. Specifically, we focus on addressing three sub research questions.

First, how to monitor whether system states satisfy a wide range of system requirements at runtime? If a requirement violation is detected by the monitor, the system controller and smart service providers can take actions to change the states, such as improving traffic performance, rejecting unsafe actions, sending alarms to police, etc. The key challenges of developing such a monitor include how to use an expressive formal language to specify system requirements so that they can be understood by machines, and how to efficiently monitor requirements that may involve multiple sensor data streams (e.g., some requirements are concerned with thousands of sensors in a smart city).

Second, how to predict a system's future states and check if the prediction satisfies system requirements? With this capability, the system operator may take actions in advance to prevent such predicted future requirement violation. A key challenge of predictive monitoring is how to account for the inherent uncertainty (e.g., due to sensor and environmental noise, unexpected events, accidents, and human behaviors) in i-CPS.

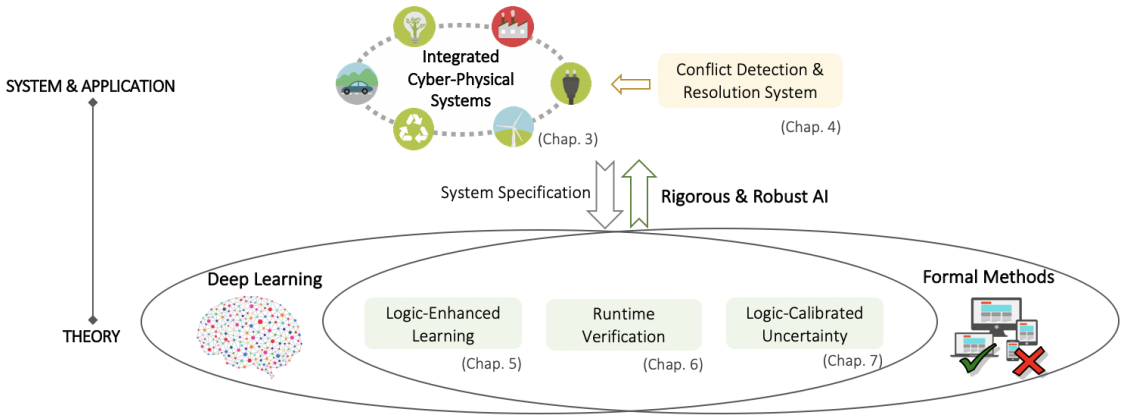


Figure 1.1: Dissertation Theme and Structure

Third, as deep learning techniques are increasingly used in i-CPS applications, how to guarantee that the deep learning results will satisfy system properties? For example, Recurrent Neural Networks have made great achievements for sequential prediction tasks in i-CPS (e.g., forecasting air quality index). Can we enforce that the learned sequence predictions must satisfy certain desired properties in i-CPS?

Terminology: Below, we first clarify the notion and scope of terms we used in this dissertation. An *integrated Cyber-Physical System (i-CPS)* is a system or platform that integrates more than one CPS operating under the same environment. A *service conflict* is defined as a situation where two or more services have actions that (1) have direct opposite effects on the same resources (called direct conflicts), or (2) have indirect cumulative effects (e.g., total mission accumulated by two services) leading to a system requirement violation (called environmental conflicts). A *system requirement* is a condition that the system is required to follow. For i-CPS, operating without conflicts is a system requirement. In this dissertation, we convert conflict detection into requirement verification (See Chapter 3 for details). System have different types of requirements, such as, real-time, safety, utilization, fairness, and robustness, which are broader than conflicts. In this dissertation, we focus on the safety and performance requirements that related to service conflicts. *Predictive monitoring* is an operation that verify the predicted future results against requirements.

Dissertation Statement: In this dissertation, we develop rigorous and robust AI for i-CPS by integrating formal methods and deep learning, as shown in Figure 1.1. The key novelty of this dissertation is using rigorous approaches based formal methods as an effective way, to bring real-world i-CPS properties and requirements into learning to verify and enhance deep learning models.

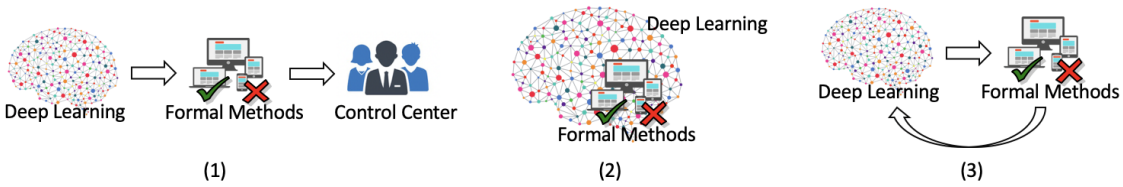


Figure 1.2: Integration of Formal Methods and Deep Learning

In particular, targeting the challenges of i-CPS, on the system and application level, we first develop a decision support conflict detection and resolution system for smart cities. On the fundamental theory level, we create three key technical innovations, i.e., prediction with logic-enhanced learning (Chapter 5), runtime verification (Chapter 6), and predictive monitoring with logic-calibrated uncertainty (Chapter 7). These techniques are general to be applied to the conflict detection and resolution system, and various i-CPS applications.

In particular, in this dissertation, we explore three approaches of integrating formal methods and deep learning. These three approaches are not mutually exclusive, i.e., multiple approaches can be applied into one comprehensive system, such as the conflict detection and resolution system we developed in Chapter 4. The first approach, as shown in Figure 1.2 (1), we apply formal methods to verify the prediction results by deep learning models and return the runtime verification results to the control center to support decisions. The conflict detection and resolution system developed in this dissertation follows this approach. It first predicts sequences of future states and verifies the predicted results against requirements at runtime. The system further controls the action of services based on the monitoring results. In the second approach, as shown in Figure 1.2 (2), we incorporate formal methods into the learning process (i.e., training phase). In this way, we verify and guide the learning process with auxiliary knowledge (See Chapter 5). The third approach, as shown in Figure 1.2 (3), we first apply formal methods to verify the learning results and then feed the results back to calibrate the learning model at the validation phase (i.e., after training phase), such as, uncertainty calibration, hyper-parameter tuning and model selection (See Chapter 7).

Next, we give a more detailed introduction of each of the five parts, followed by a summary of contributions and impacts.

1.1 Integrated Cyber-Physical Systems

An important step of bringing integrated formal methods and machine learning techniques to real-world applications is to systematically learn from large scale real-world data and applications. In this dissertation, we conducted a series of data-driven analytic as follows. (1) In order to identify the desirable features to have in a specification language for cities, we systematically studied and annotated over 1000 city requirements (e.g., standards, regulations) across different domains, including energy, environment, transportation and public safety from more than 75 cities around the world, which is the first study of city requirements in the field. (2) We identified key types of service conflicts, model properties, and uncertainty by analyzing large-scale cross-domain city data. (3) We also developed simulations with implemented services to evaluate the solutions in smart cities. (4) Finally, we published new data sets online. The analytic results identify real-world research questions and support the development of new methods and systems.

1.2 Conflict Detection and Resolution

Researchers have accumulated abundant knowledge on how to design AI-powered services independently. However, underlying expected or unexpected couplings among services due to complex interactions of social and physical activities are under-explored, which lead to potential conflicts, including direct device conflicts and environmental conflicts. We develop the first system for conflict detection and resolution in i-CPS, especially for in smart cities. We identify conflicts in large-scale i-CPS and formalize i-CPS safety and performance requirements using Signal Temporal Logic (STL). By verifying predicted sequences with formal specified requirements, our solution deals with dynamic conflicts that cannot be apriori detected. Using formal methods to generate and verify resolution options, the system provides sophisticated information (e.g., quantitative trade-offs, confidence levels) to support human decision making.

This system significantly improves city's safety and performance in experiments and simulations. Meanwhile, we are also working on deploying this system in Newark City, NJ.

1.3 Logic-Enhanced Learning

Recurrent Neural Networks have outstanding achievements for prediction and decision-making support for CPS. However, for large-scale and complex integrated CPS, RNN models are not always robust,

often subject to anomalies and uncertainty, especially when the predictions are projected into the future (errors grow over time). Real-world systems often follow certain model properties, which cannot be guaranteed by the existing prediction models. In this dissertation, a formal logic enhanced learning framework with logic-based criteria to enhance RNN models to follow system critical properties. The novelty of this framework is incorporating the system critical properties into the learning process in an end-to-end manner with back-propagation. This framework is general and can be applied to various RNN models. The evaluation results on large-scale real-world city datasets showed that this work not only improved the accuracy of predictions, but importantly also guarantees the satisfaction of model properties and increases the robustness of RNNs.

The proposed STLnet is broadly applicable to various sequential prediction tasks beyond smart cities. This work shows the promise of leveraging formal methods to enhance the robustness and reliability of deep learning.

1.4 Runtime Verification

We create novel formal specification languages and efficient runtime verification techniques for large-scale i-CPS with theoretical guarantees. To tackle the limitation of existing formal logic, we develop SaSTL—a novel Spatial Aggregation Signal Temporal Logic—for the efficient runtime monitoring of safety and performance requirements. The new specification languages achieve over 95% expressiveness coverage on real-world city requirements while the state-of-the-art only achieves 43%. Additionally, we build a SaSTL-based monitoring tool to support specifying and runtime monitoring requirements by different stakeholders in smart cities.

1.5 Logic-Calibrated Uncertainty

We develop a novel approach for monitoring sequential predictions generated from Bayesian Recurrent Neural Networks (RNNs) that can capture the inherent uncertainty in CPS, drawing on insights from our study of real-world CPS datasets. We propose a new logic named Signal Temporal Logic with Uncertainty (STL-U) to monitor a flowpipe containing an infinite set of uncertain sequences predicted by Bayesian RNNs. Furthermore, we develop novel criteria that leverage STL-U monitoring results to calibrate the uncertainty estimation in Bayesian RNNs. Evaluation results on large-scale real-world city data show that our approaches significantly improve the accuracy and robustness of

deep learning models and achieve well-calibrated uncertainty. Moreover, the system also effectively improves smart cities' safety and performance in smart city simulations.

The STL-U predictive monitoring approach demonstrates the feasibility of integrating formal methods and Bayesian deep learning for the predictive monitoring of safety and performance requirements in smart cities. In addition, the proposed STL-U criteria can be applied for the uncertainty estimation in a wide range of deep learning applications. Compared with traditional uncertainty estimation methods, the proposed logic-based solution can lead to better uncertainty calibration for sequential prediction tasks.

1.6 Summary of Contributions

In this dissertation, we develop rigorous and robust models for reliable i-CPS by integrating formal methods and deep learning with the following main contributions.

- Conducting data-driven analytic on large-scale data to identify key features of i-CPS.
- Developing a decision support system for conflict detection and resolution among integrated IoT services in i-CPS.
- Presenting novel formal specification languages and efficient runtime verification techniques for large-scale i-CPS with theoretical guarantees.
- Building novel formal methods enhanced deep learning techniques to increase the robustness of sequential prediction by incorporating formal specification and verification into the learning process.
- Creating a novel approach for monitoring sequential predictions generated from Bayesian Recurrent Neural Networks and leverage predictive monitoring results to calibrate the uncertainty estimation in Bayesian RNNs.

1.7 Dissertation Structure

We show the main structure of this dissertation in Figure 1.1. In the rest of this dissertation, we first discuss the related work and how this dissertation advances the state-of-the-arts research in Chapter 2. In Chapter 3, we present the basic concepts of integrated Cyber-Physical Systems, where

we formally define conflicts and analyze their effects. Next, we present a decision support system for conflict detection and resolution in Chapter 4. We develop a logic enhanced learning framework for prediction in Chapter 5, create a novel specification language for i-CPS monitoring in Chapter 6, and build a predictive monitoring approach with logic-calibrated uncertainty in Chapter 7. Finally, we summarize the dissertation and discuss its broader impact in Chapter 8.

Chapter 2

Related Work

2.1 Integrated Cyber-Physical Systems

2.1.1 IoT Platforms for Smart Cities

Given advances in the Internet of Things (IoT) and AI, existing city Operations Centers deploy various sensing and control platforms to monitor city states and services for decision support. There are several commercially available IoT platforms for smart cities, such as, IBM Watson IoT [3], Azure IoT suite from Microsoft [4], Intel IoT platform [5], and AWS IoT from Amazon [6]. They provide support for setting up IoT infrastructure customized to application requirements. They address different aspects of potential city-level IoT infrastructure, including but not limited to, scalability of sensing and actuator modules, real-time response, cloud support for IoT, real-time stream analytics, raw data storage, data driven dynamic applications, and network and data security. Although such IoT platforms can be utilized for developing scalable smart city services, they consider smart city applications as independent entities. Hence, they don't focus on the integration of services and its subsequent complexity (e.g., concurrency and conflicts). While the existing IoT platforms provide support for safeguarding connected devices, networks, data transmissions and data accessibility of the IoT infrastructure, none of them addresses the safety challenges introduced by integration of systems/services (e.g., conflicting operations, policy violations, and conflicting effects of services). These decision support systems only focus on data visualization and analytics based on city status statistics, IoT management, task assignment, and do not have the capability of detecting and resolving

service conflicts, which require sophisticated reasoning, prediction, and intervention. To the best of our knowledge, we are the first to formulate and develop a safety-aware system for detecting and resolving potential conflicts in the context of smart cities.

2.1.2 Safety in Automation Systems

Safety issues have been well studied in automation systems. Standards and rules for functional safety of automation systems have been made, such as IEC 61508 and ISO 26262 [7], which define functional safety for automotive equipment applicable throughout the lifecycle of all automotive electronic and electrical safety-related systems. They support the product development from hardware, software and system levels, and provide safety analysis. These standards can be very helpful when extended to the development of single smart services in smart cities. However, there are no rules for interactions or conflicts among services/systems.

There is some research on the functional safety among multi-agents in automation systems, such as building automation and control systems. The authors in [8] focus on the functions affecting peoples' safety, security and health while maintaining the functional safety and system security of both the network nodes and the communication protocols. Resendes et. al. present a survey on the conflict detection and resolution in home and building automation systems [9]. Pallottino et al. propose a cooperative policy for conflict resolution in multi-vehicle systems, which rests on the assumption that all agents are cooperating by implementing the same traffic rules [10]. These integrated systems are only within the domains of transportation, homes, and buildings. However, functional safety in smart cities is much more complicated as it involve multiple domains, different types of services and a significantly large number of actuators.

2.1.3 City Simulator

SUMO [11] is an open source microscopic traffic flow simulation, which can import net/map and traffic demand modeling components. It has been utilized in several traffic flow related research, such as vehicular communication, route choice and dynamic navigation traffic light algorithms, emission and noise modeling, and person-based Intermodal traffic simulation In our work, SUMO is used as a smart city simulator with the help of the Traffic Control Interface (TraCI) [12], a technique for interlinking road traffic and network simulators. With TraCI, the behavior of vehicles during simulation runtime can be controlled and thus the influence of actions on the simulated city are better understood.

2.2 Conflict Detection and Resolution

2.2.1 Detection

There are some existing systems to detect dependencies across multiple human centric CPS systems. Munir et al. focus on detecting dependencies across interventions generated by different human-in-the-loop apps (e.g., health apps) [13]. They use simulated apps and structured metadata from each app. Metadata contain (i) interventions performed by each app and (ii) corresponding potential physiological parameters that might be affected by each intervention. They rely on a physiological simulator [14] to approximate potential effects of an intervention. Preum et al. developed Preclude, a system to detect conflicts in textual health advice generated from smart phone health applications and health websites [15]. While they provide a taxonomy of conflicts in health advice and solution to detect different types of conflicts, their approach is focused on textual interventions only.

Another relevant system is DepSys that aims at detecting dependencies from multiple smart home apps [16]. It is a utility sensing and actuation infrastructure specifically designed for smart homes that detects, and resolves conflicts among multiple smart home apps by addressing multiple dependencies. Although our work is similar to DepSys in terms of the end goal (i.e., conflict detection), one fundamental difference lies in the underlying assumption about meta data. DepSys solely depends on the meta data of apps provided by the app developers. HomeOS [17] is a PC abstraction to improve manageability and extensibility for smart home apps. It exposes services to home app developers with simple abstractions to access home devices and allows easy incorporation of home devices and applications using common protocols (e.g., Z-Wave and DLNA) and many kinds of devices (e.g., lights, media renderers and door/window sensors). SIFT [18], a safety centric programming platform, detects whether apps running in an IoT environment conform to safety policies and whether apps result in logical conflict with each other. Although it detects logical conflicts (equivalent to *opposite* conflict of CityGuard) using rule based approach, it does not consider different nuances of detecting conflicts / policy violations, e.g., effects, secondary effects, emphasis, and conditions, and it is not applied to inter-services problems.

2.2.2 Resolution

Researchers also propose different ways for conflict resolution based on its conflict types [9]. For example, for interest conflict, [19] considers the application's demands for quality of services and resource consumption, and uses a client-server architecture model to select the conflict resolution,

which, however, can only deal with very small scale clients. [20] builds a resource management method to resolve the conflict in smart buildings. [21] builds an ontology-based policy framework using an AI planner to detect and automatically resolve policy conflict. However, these methods focus on detecting and resolving direct conflicts. They do not (1) consider the secondary effects on the environment, which are where most of city conflicts arise, or (2) predict and avoid conflicts. The environment of smart cities is so complicated that the conflicts among smart services not only include all the above conflict types, but also contain different types of environmental conflicts. Rule-based or resource management approaches are not sophisticated enough to predict and prevent conflicts in smart cities. Our approach uses a feedback control loop which monitors city states in real time, and predicts the effects of actions and the potential resolution options by predicting and verifying future city traces.

2.3 Logic-Enhanced Learning

The attempts of enforcing machine learning to follow logic rules are dated to the early stage of the development of the neural network. So-called Neural Symbolic Systems [22, 23] construct network architectures to combine inference with logic rules. Combining logic rules with various machine learning models has been successful [24, 25]. Breaking the black box of the neural network has always been a popular research topic. Applying logic rules, as one typical approach to break the black box, has attracted much attention. A direct solution is to formulate the logic rule as an optimized loss item. By minimizing the logic loss, soft constraints of the logic rule are proposed to the model [26, 27, 28]. Other methods include Logistic circuits [29] and Logic Tensor Network [30] design specific structural to incorporate logic rules. [31] generates a graph model to embed logic rule into the prediction. Following knowledge distillation [32], [33] proposes a way to integrate rules defined by first-order logic with knowledge distillation. Following this method, several works [34, 35] propose ways to better train deep NLP models with some specific type of logic rules, which uses posterior regularization to constraint the student network. However, most previous works only apply simple and straightforward logic rules, target single variable classification problems (e.g., sentiment classification), and only apply a soft constraint to (rather than a guarantee) the satisfaction. Different from previous work, our work targets multivariate sequential prediction models and guides them to learn the model properties with complex temporal features in regression tasks. Therefore, we chose *Signal Temporal Logic*, a logic variant that focuses on the temporal properties, to formalize the model properties. As a powerful specification language, STL has been broadly applied to the

model checking, specification and verification for CPS applications [36], such as robotics [37], smart cities [38, 39], healthcare [40]. The introduction of temporal operators (e.g., always, eventually and until) makes STL more natural, intuitive and flexible in describing dynamic systems.

2.4 Runtime Verification

Monitoring spatial-temporal properties over CPS executions has been initially investigated in [41, 42], where the authors introduced a spatial-temporal event-based model for monitoring CPS. In this model, events are labeled with time and space stamps. These events can be triggered by actions, exchange of messages or physical changes. A centralized monitor is then responsible to process all these events. Their approach provides an algorithmic framework enabling a user to manually develop a monitor, but they do not provide any spatial-temporal specification language. The literature instead offers several logic-based specification languages to reason about the spatial structure of concurrent systems [43], medical images [44], and the topological [45] or directional [46] aspects of the interacting components. However, these logics are not practical for monitoring CPS, because they are generally computationally complex [46] or even undecidable [47].

Specification-based monitoring of spatial-temporal properties over CPS executions has become practical only recently with SpaTeL [48] and SSTL [49]. SpaTeL extends the Signal Temporal Logic [50] (STL) with the Tree Spatial Superposition Logic (TSSL) [51, 52]. TSSL classifies and detects spatial patterns by reasoning over-quad trees, suitable spatial data structures that are constructed by recursively partitioning the space into uniform quadrants. The notion of superposition in TSSL [52] provides a way to describe statistically the distribution of discrete states in a particular partition of the space and the spatial operators corresponding to *zooming in and out in* a particular region of the space. By nesting these operators, it is possible to specify self-similar and fractal-like structures [53] that generally characterize the patterns emerging in nature such as the electrical spiral formation in cardiac tissues [54]. The procedure allows one to capture very complex spatial structures, but at the price of a complex formulation of spatial properties, which are in practice only learned from some template image.

SSTL [49] extends STL with several spatial operators (i.e., somewhere, everywhere, and surround). The SSTL semantics operates on a weighted undirected graph, where the weight on each edge represents the distance between two nodes. The Spatial Temporal Reach and Escape Logic (STREL) [55, 56] generalizes SSTL, by introducing two new spatial operators, (*reach* and *escape*), which are able to

express the same spatial operators of SSTL. Furthermore, while SSTL can be applied only on static weight undirected graphs, STREL can be applied also to dynamic networks. However, both SSTL and STREL do not support spatial aggregation operators that we show to be an important feature for monitoring smart cities.

2.5 Logic-Calibrated Uncertainty

2.5.1 Predictive monitoring for CPS

The research area of predictive monitoring has been drawing increasing attention in recent years. For example, (Bayesian) Neural Predictive Monitoring [57, 58] checks predictions about neural state classification and uses a principled criteria to reject predictions that are likely to be incorrect; a predictive monitor for rare failures is developed in [59] using Discrete-Time Markov Chains trained with samples of rare events; STLnet [60] incorporates predictive monitoring into the learning process and enhance RNN-based sequential prediction models to follow STL specified model properties in both training and testing processes. Prevent [61] and other runtime verification techniques with state estimation [62, 63, 64] use Hidden Markov Models or Dynamic Bayesian Networks to learn and predict the probability of a hidden state satisfying a safety property. A more recent approach [65] uses instead linear hybrid models of the system under monitoring to bound the uncertainty in the gaps between consecutive samples.

These existing works mostly focus on monitoring individual predictions rather than sequential predictions. A more recent work [66] applies statistical time-series analysis techniques (e.g., ARIMA) to forecast future signal values and computes the satisfaction probability of a STL formula over the prediction horizon; however, the applicability of this approach is limited by the assumption that a joint probability distribution of predictions over multiple time-points can be estimated. By contrast, our approach considers uncertain sequential predictions generated by Bayesian RNN models, which are generally applicable to many CPS domains.

2.5.2 Temporal logic based runtime monitoring with uncertainty

Over the past decades, tremendous progress has been made in developing techniques and tools of runtime monitoring (also called runtime verification) based on rigorous specifications expressed in various temporal logics (e.g., LTL, STL). For example, a survey on STL-based runtime monitoring for CPS is provided in [67], which includes applications such as automotive systems and medical

devices; a STL-based framework is developed in [38] for detecting requirement violations in smart cities; SaSTL [39] extends STL for runtime monitoring of spatial-temporal properties in CPS; and another spatial-temporal logic named SpaTeL is applied to monitor the power grid in [68]. However, most of the literature focuses on monitoring deterministic multi-variable signals, which is a limiting factor when we need to monitor predictive models and to reason about uncertainty.

There are some attempts to handle uncertainty by incorporating random variables in predicates. For example, C2TL [69] checks the probability of a deterministic signal satisfying a linear constraint whose coefficients are random variables; PrSTL [70] uses atomic predicates that are parameterized with a time-varying random variable over a deterministic signal; StSTL [71] checks the probability of a real-valued measurable function over stochastic signals; and StTL [72] extends StSTL to reason about the robustness of requirement satisfaction. Our approach differs from these previous works in several aspects. First, the proposed STL-U monitor checks a flowpipe signal that contains an infinite set of uncertain sequences rather than a single sequence. Moreover, instead of computing a single probability value of satisfying a predicate, STL-U reasons about the uncertainty captured by confidence intervals of the flowpipe signal, which is a more suitable representation of uncertainty estimated from Bayesian deep learning.

The problem of monitoring an infinite set of sequences has been studied before in [73] for the reachability analysis of continuous and hybrid system models. This work proposes a Reachset Temporal Logic (RTL), which extends STL and is defined on the reach sequence (i.e., a function mapping time to the set of states reachable from a set of initial states and uncertain inputs). The notion of reachability in RTL (i.e., checking if all the values within the reach sequence satisfies a formula) can be encoded as STL-U strong satisfaction with our approach. In RTL, the formulas are in positive normal form: the negation operator can appear only in basic propositions while it remains undefined for generic formulas that may include also temporal operators. Furthermore, the RTL-based model checking algorithm is limited to a specific fragment of RTL where the only possible temporal operator that can be used is the *next* operator. Similar to RTL [73], the parameter synthesis approach presented in [74] introduces a new semantics for the positive normal form fragment of STL that is defined on sets of traces rather than on a single trace.

In our approach, we introduce the notion of strong/weak semantics to handle the negation operator with more generic formulas: we define the evaluation of strong satisfaction for a formula $\neg\varphi$ (for both atomic predicates and temporal formulas) as equivalent to the violation of the weak satisfaction for

the formula φ and vice-versa. Our work takes inspiration from the paper of Eisner et al. [75] where the authors propose a weak/strong semantics to reason with linear temporal logic (LTL) on truncated paths: the weak semantics provides an optimistic view of the satisfaction of an LTL formula on a truncated path, while the strong semantics provides a pessimistic view. In our approach instead, the weak/strong semantics is used to change the existential/universal quantifier when we interpret a proposition over a confidence interval.

2.5.3 Uncertainty estimation in deep learning

While most deep learning models do not offer the uncertainty of their predictions [76], works that capture the uncertainty (or confidence) of the prediction can be dated to the early development of neural networks in the 90s'. Bayesian Neural Network [77] represents a probabilistic model that infers a distribution as output. Bayesian Neural Network is known to be robust and resilient to overfitting. However, the hardness of inference prevents the prevalence of the model in practice. Following these directions, several works [78, 79] use variational inference to perform an approximated inference on Bayesian Neural Networks. Aside from variational inference, Monte Carlo Dropout is another approach to obtain uncertainty estimation of the model [80, 81]. By exploiting the dropout structure in the deep neural network, these approaches turn the original Neural Network model into a simple Bayesian Neural Network without changing the structure and apply approximated inference with the Monte Carlo approach. Existing works [82, 81, 83] mostly focus uncertain estimation on single-time classification or regression tasks. This dissertation focuses on the case of time series prediction. Moreover, in contrast to previous measures of uncertainty that are rather empirical, our work proposes a formal framework to model and define requirements to the output distribution. Our work can thus be used to provide a confidence guarantee of the model prediction and to evaluate the quality of the uncertainty estimation.

Chapter 3

Integrated Cyber-Physical Systems

With the arrival of technological tools such the Internet of Things (IoT), Big Data, Cloud computing products [84], and Crowd Sourcing platforms, cities are becoming increasingly able to monitor the state of their infrastructure, services, and populace, cost effectively and at scale. With a connected populace and infrastructure, cities are also able to dynamically act on changes with increased accuracy based on the observations it makes. A city that employs such technologies to improve the quality of life for it inhabitants, and the competitiveness of it's economy is commonly referred to as a smart city. There are a number of cities that are already embracing the notion of a smart city, such as the city of Santander in Spain [85].

One key open problem is that with many services operating simultaneously, conflicts will arise, as shown in Figure 3.1. Conflicts have both an immediate effect on human life, as well as long term secondary effects. In the complex system-of-systems that is a smart city, services will come into conflict when contending over the same resources, incurring opposing actions, and when having contradictory or conflicting objectives. These conflicts are both institutional and technical, and have to be resolved holistically. Finding and classifying conflicts are non-trivial, but crucial to the operation of a smart city due to (i) the scale of a smart city system, (ii) the diversity in services, and (iii) the wide range of ways the services interact with the city. While some of these conflicts can be detected during the design phase of services, many conflicts can occur unpredictably at runtime, i.e., when the implementation phase is over and the services are operating simultaneously. Detecting runtime conflicts is significantly more challenging than detecting design time conflicts, as runtime

conflicts involve a higher degree of uncertainty. Some conflicts can be identified a priori, but most will occur unpredictably at run-time. Once a runtime conflict is identified, resolving it often involves a compromise, such trade offs have both a technical and administrative component.

This chapter primarily explores the nature of the runtime conflicts that arise in a smart city. The primary contributions of this chapter are:

- The enumeration of smart city services characteristics.
- The classification of conflicts.
- An evaluation of conflict analysis that demonstrates the high probability of conflicts using actual data from a smart city.
- An evaluation of conflicts using services that shows that the high probability of conflicts can be expected to grow in the future and that many conflicts can only be resolved at runtime.

3.1 Smart City Services

A smart city is a system of systems, where each system represents a specific domain (e.g., transportation, public safety, utility, emergency, environment, city planning and operations) and each domain consists of a set of services. For example, the public safety domain may include police patrolling services, traffic violation control services, and road accident management services, etc. Similarly, the transportation domain includes public transport services, road work services, etc. Each service performs a set of functions to fulfill an objective, e.g., a traffic violation control service penalizes drivers for speeding. The functions may be triggered by an event (e.g., a traffic violation) or scheduled statically (e.g., turning off street lights at dawn to save energy). Functions may produce a set of effects upon completion, e.g., blocking a lane for road work. Effects that are directly actuated by a service are primary effects. Effects that are the outcomes of a primary effect are characterized as secondary effects. Thus a single action can create a chain of subsequent effects.

Potential services from different domains are described in Table 3.1. Although not exhaustive, typical services that covers several major functions of a smart city are included. Most of these services either exist already in some cities or are going to be implemented in the near future. How such essential services frequently conflict with each other when they run simultaneously will be demonstrated in (Section 3.5).

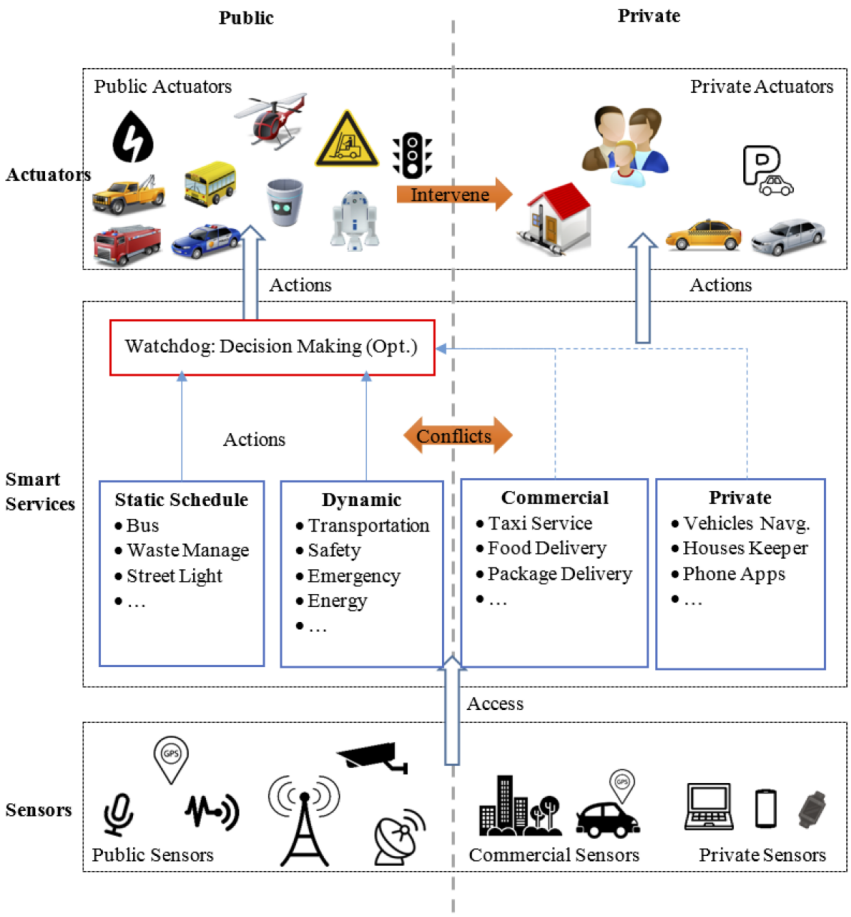


Figure 3.1: The Integrated System of Smart Cities

Table 3.1: A List of Services in Smart Cities

ID	Services	Domain	Description
1	Street Lights Control [86]	Environment	It controls level of illumination in city streets by 1) turning on/ turning off street lights and 2) adjusting brightness of street lights according to ambiance.
2	Street Robots Management	Environment	Robots on the street sense different environment states (e.g., weather, light, pollution level) and aid a passer by if necessary (e.g., kid, disabled people).
3	Waste Management [84]	Environment	It performs waste collection, disposal, recycling, and recovery. It sends out garbage collection trucks regularly and extra ones when the containers are over 2/3 full.
4	Delivery Management	Environment	It maintains a dynamic schedule of package delivery trucks based on real-time demand.
5	VIP Delivery Management	Environment	It dispatches drones to carry packages to customers.
6	Air Pollution Control [85]	Environment	When air pollution is detected to cross the safety threshold, it will 1) send out personalized sms to citizens based on location and physiological state, 2) post messages on street screen, and 3) suggest authorities to determining the cause of pollution and reduce vehicles on the streets.
7	Noise Pollution Control [85]	Environment	If noise pollution is detected to cross the safety threshold, it will 1) send out sms to citizens, 2) post messages on street screen, and 3) turn off public speakers/alarms.

8	Port Pollution Control	Environment	It detects potential pollution caused by incoming ships and stops polluting ones from coming to the port. When serious pollution is detected, it will send out sms to nearby ship and stop them from coming.
9	Culture Event Management [85]	Environment	It facilitates the diffusion of information about cultural activities and motivates people to be involved in them. It also helps to manage the facilities (parking, lights) and around neighborhoods during cultural events.
10	Live News Gathering [87]	Environment	It manages navigation of autonomous news coverage vans and drones to breaking news sites for capturing video/photographs.
11	Adaptive Traffic Light	Transportation	It adjusts signal lights dynamically based on traffic density to maximize utilization and prevent traffic congestion.
12	Emergency Vehicle Monitor	Transportation	It monitors streets and adjusts traffic lights to minimize delay of emergency vehicles, e.g., police cars, ambulances, firetrucks.
13	Road Condition Monitor	Transportation	Upon detecting light snow/rain, it sends alarm to nearby drivers. Upon detecting heavy snow/rain/flood, it adjusts signal lights to block the road and reroute vehicles.
14	VIP Route Scheduling	Transportation	It reroutes regular vehicles off VIP routes and programs the shortest path for VIP cars.
15	Traffic on Special Events	Transportation	To accommodate visitors during games/concerts/other events, it blocks some streets and adjusts traffic signals on event days.
16	Road Work Service	Transportation	It manages road works and road side constructions. It reroutes vehicles to alternate paths when road work is going on.
17	Smart Parking [88]	Transportation	The system informs drivers about the number of available parking spaces in adjacent areas and gives direction to desired parking lot.
18	Bus Schedule Service [89]	Transportation	It manages the bus schedule both statically and dynamically. Based on passenger demand, it reduces bus interval/waiting time and directly sends extra buses to some bus stops.
19	Taxi Dispatch Service [89]	Transportation	In case of increase in demand, it sends extra taxis to corresponding event locations.
20	Traffic Violation Control	Public Safety	It pulls over vehicles for traffic violations.
21	Road Accident Management [87]	Public Safety	In case of a road accident it 1) notifies law enforcement services and GPS navigation services, 2) blocks roads temporarily, 3) sends message to vehicles and street screens, and 4) adjusts traffic lights to regulate traffic flows and prevent traffic jams.
22	Risky Area Monitor [89]	Public Safety	It co-ordinates sensors (i.e., cameras, street lights) and actuators for real-time monitoring of risky areas. Upon detecting any crime or police intervention, it alerts citizens to avoid such areas temporarily.
23	Raiding Crime Scenes	Public Safety	It conducts raiding operation in crime scenes/risky zones. During the raid, it can block roads without any prior notice if necessary.
24	Destroying Obsolete Structures	Public Safety	It blocks some roads temporarily when blowing up any obsolete structures: bridge/buildings. It informs residents in nearby areas ahead of time and and blocks off nearby roads during the operation. Informing adjacent vehicles about the event involves uncertainty.
25	Potential Terrorist Attack Monitor	Public Safety	When it detects a potential terror threat / attack it 1) postpones operation of trains/public transports in concerned areas, 2) re-routes vehicles, and 3) often uses bio-chemical weapons against potential threat(s).
26	Surveillance Drone Management	Public Safety	It uses drones to monitor safety conditions and detects potential threats over streets and buildings.
27	Public Security	Public Safety	It helps public organizations and houses to protect citizens' goods and feeds real-time information to fire and police departments when detects an intrusion or theft.
28	Fire/Explosion Management	Emergency	It detects and automatically takes action based on the level of severity, such as: 1) detecting false alarms, 2) informing firefighters and ambulance, 3) blocking off nearby streets/buildings if necessary, 4) helping people to evacuate, and 5) co-ordinating rescue drones and robots.
29	Inclement Weather Alert	Emergency	It alerts and gives personalized advice on how to stay safe during emergency (storm, earthquake, tsunami,flood) through messages, suggest the car to stop upon detecting earthquake, manages street lights and other utilities safely.

30	Evacuation Aid	Emergency	It helps people evacuate during extreme emergencies. Specifically, it detects the location of people and sends rescue, map, message of instruction to the phones, and manages drones and robots for rescue service.
31	Automatic Health-Care Dispatch	Emergency	It provides 24/7 health care for patients. When it detects an emergent situation of patient, it will sends an ambulance or helicopter to pick up the patients and send them to the most suitable hospital.
32	Ambulance Management	Emergency	It sends ambulance to help patients and send them to the nearest hospital when someone calls an ambulance.
33	Water Pipe Monitor	Emergency	Upon detecting pipe leakage, it turns off the water flow and sends service crew. It blocks street around if necessary (e.g. manhole concerned).
34	Gas Pipe Monitor	Emergency	It monitors gas pipe, and alerts and evacuates people around based on different degree of severity, cuts off electrical utilities if necessary and sends service crew upon detecting the gas leakage.
35	Electricity Monitor	Emergency	It starts the back-up generator, and sends crew upon detecting any technical fault. It also sends alert to adjacent people and take proper intervention.
36	Gun-Shot Detection	Emergency	It alerts nearby patrol police immediately and sends messages to people in the same neighborhood. It also analyzes surveillance (if any) to find potential suspects.
37	Network Error Detection	Emergency	When detects network error, it informs people immediately. After network error, it exams all services if they are functioning properly or not.
38	Sleep Mode for City	Emergency	It turns the city into sleep mode gradually if serious emergency arises (utility breakdown, network failure etc.) while guaranteeing the basic functionality of city.
39	Water Usage Monitor [85]	Energy	It monitors water usage and turns off water flow for a short period if excessive water is consumed continuously for a long duration.
40	Energy Usage Monitor [85]	Energy	It turns off electrical devices at idle hours to save power.
41	Solar Energy Generation Optimization	Energy	When solar energy is available, it turns machines to solar energy mode.

3.1.1 Characteristics of Smart City Services

The characteristics of a smart city service describe how it interacts with the city's resources and other services, how it affects the environment and people, and what requirements it imposes on the infrastructure it employs, such as sensors and actuators. Below, some fundamental characteristics of smart city services and their interaction with each other are introduced.

Uncertainty: It refers to unforeseen events or processes that can not be accounted for by the services ahead of time or are unknown to the service but affect its performance. Uncertainty can occur in the different layers of a service: sensing layer (a pollution sensor attached with a vehicle moved from its expected location), communication layer (network failure), or actuation layer (control valve only opens partially). Uncertainty usually arises in the flow of information and/or resource availability. Uncertainties can cause change(s) in the course of action. This characteristic emphasizes the need to design an integrated service platform.

Dynamism: Although some services of a smart city can operate with a static schedule (e.g., sending out a garbage truck every morning), a major portion of services function dynamically. For example, public transport service schedules bus routes and frequency based on demand: more buses when there is a festival/game/concert. Such dynamic operation of a service can create complexity if the service shares resources (sensors/actuators/data) with other service(s).

Real-time: In a smart city, services frequently rely on real-time information for operational decision making. One of the most common smart city service is monitoring crime-prone areas through surveillance cameras and mitigating risky incidents. It relies heavily on real-time video feed.

Mobility/Spatio-temporal availability: Often services involve mobility in terms of sensing and actuation, e.g., police patrol service and garbage collection service. This poses a new set of challenges, including, coverage, redundancy, operational cost, scheduling, communication between the control and physical layers. Also, mobility contributes to uncertainty, and the degree of dynamism.

Duration and scale of effect: Any function performed by a service results in effects or a chain of effects into the future. These effects vary in duration. For example, blocking a road can affect traffic for a long time. Some services require large and/or lengthy actuation to make small changes, while others require small and/or short actuation to make significant changes. This complicates the control mechanism. In addition, the notion of duration and scale of effect can lead to uncertainty, real-time feedback, and resource constraint.

Efficiency: Service behavior is determined by the targeted efficiency. Efficiency can be measured as a function of resources, cost, and time. Targeted efficiency can control the number of sensors and actuators used in a function of a service, e.g., increasing the number of buses to meet public demand on weekdays. Maximizing efficiency for one service can often lead to resource constraints for another service if the two services share any resource. Thus it poses an optimization problem with constraints on resources and operational costs.

Ownership: A service can be private, public, or commercial in terms of ownership. The degree of interaction and information flow between services with different ownerships can vary according to service design, and city policies.

Although not completely, the characteristics above outline the potential **complexity** of smart city services. Moreover, the characteristics play a vital role in creating the context of potential **conflicts** among the services as described in Section 3.2. It is acknowledged that these characteristics affect

one another and cannot be quantified on their own. For example, mobility and efficiency affect uncertainty.

3.1.2 Safety Requirements

Coverage: City safety generally involves safety for the environment and humans. *Environmental Safety* includes maintaining the quality of air, water, noise, and weather and safety of property. For example, actions taken by a service should not result in air, water, or noise pollution, or street lights should be kept on at night in crime-prone areas. *Human Safety* refers to protecting citizens from any dangerous or unhealthy situation (e.g., road accidents).

Consequently, the overall *Smart System Safety* means that all the smart services running in a city must neither bring danger to the environment or citizens, nor conflict with each other. For example, autonomous vehicles shall not hit pedestrians or cause environmental damage. Also, a smart traffic service and a smart emergency service should not try to turn the same traffic light to green and red simultaneously.

Context: In the complex and dynamic setting of real time smart cities, safety and performance requirements need to be aware of context, i.e., they need to accommodate special circumstances, because rigid/static safety requirements can result in unwarranted / catastrophic consequences. For example, assume a transportation domain service sets the highest traffic capacity of a street, *street A* to 100 to avoid congestion (i.e., a performance metric) although the street may accommodate more vehicles for a short time. However, when there is a fire on a nearby street, *street B* vehicles may have to evacuate through *street A* (i.e., for safety). In this circumstance, the performance requirement of maintaining traffic capacity in *street A* should be aware of the safety requirement of *street B*. Thus the safety and performance requirements should be context aware.

Emphasis: As another complication, range and frequency of temporal and spatial entities should also be considered when specifying unsafe situations. For example, the traffic congestion lasting for 5 minutes might not be considered as an unsafe situation, but the one lasting for 1 hour is unsafe.

3.1.3 Effects of Actions from Services

Primary and Secondary Effects: After an action is taken, it has a series of effects on the city. A primary effect relates to the main purpose of the action. For example, to control the noise level in

a school area, the noise control service does not allow trucks to go through the school area during the day and redirects them to a nearby residential area. The primary effect of this action is the reduction of the noise level in the school area during day time. However, this action may result in one or more secondary effects. For example, in this case, the traffic volume of the nearby residential area may increase as some traffic is redirected from the school area. Such secondary effects may have a serious influence on the city environment, which may violate the city safety requirements.

Spatial and Temporal Range: Most effects of actions are not limited to a single location at a single time. Instead, the effects have *spatial and temporal* ranges of influence, which need to be considered when detecting conflicts. Following the above example, since trucks have to drive on other roads when passing through this area, the trucks will have an effect on the traffic on these roads and the added traffic may cause congestion (i.e., a violation of a performance requirement) for several hours. Also, an increased volume of trucks on a particular day may result in increased release of air pollutants causing air pollution (i.e., a violation of a safety requirement).

3.1.4 Integration of Smart City Services

Service integration is integral in the context of a smart city for the following reasons.

First, a service might frequently interact with other services from the same domain as well as from other domains. For example, when there is a road accident, both the road accident service and the emergency dispatch service usually act together to address the situation. Service interaction can occur at different degrees based on the corresponding situation. To make such interactions functional, effective, and efficient, services must be integrated. The integration process must start from as early as the design phase of the services.

Second, some scenarios in a smart city require services to share resources both in sensing and actuation layers. For instance, the same set of trucks can be shared for collecting garbage and carrying air pollution sensors. An integrated service platform that is aware of the demands, constraints, and objectives of each service, can contribute to efficient scheduling of shared resources. Thus service integration is vital for quality of services (QoS) and operational efficiency.

Third, services are often correlated with each other in terms of sequence of operations, i.e., output of one service is the input to another service. For example, a road accident can cause traffic delay. If a GPS enabled navigation service (e.g., Google maps) is not aware of the accident, it may yield an

Table 3.2: Examples of conflicts of services in smart cities

Category	Type	Example
Device	Opposite	Pedestrian service turn a traffic signal to green, while Traffic Congestion Service turn the same traffic signal to red.
Device	Numeric	Traffic Service set the speed of autonomous vehicles to be 70 mph; Safety Service set the speed of them to be 60 mph.
Device	Duration	Emergency service keeps the traffic lights *green* for 10 minutes to allow ambulances to move faster while traffic congestion service needs it to turn *red* every 2 minutes.
Environment	Single	Air quality control service redirect the traffic to reduce air pollution, but cause serious traffic congestion on the other road, level of which exceeds safety requirement.
Environment	Opposite	While emergency service tries to evacuate an area, traffic congestion service directs more vehicles there.
Environment	Additive	Event service caused a certain level of noise below threshold, emergency service caused a level of noise below threshold, but the additive level is above threshold.
Environment	Dependent	Traffic service can only direct vehicles to street 1 after the water pipe leak is resolved by emergency service.

erroneous estimate of the route time. Services must be integrated to ensure proper data flow among them.

3.2 Conflicts in Services

This section describes how the different characteristics of smart city services contribute to conflict. Specifically, it presents potential sources of conflicts and enumerates various types of conflicts.

Broadly speaking, conflicts in services arise when the actions from two services can not be performed together without adverse effects. Conflicts can occur at devices, in the environment or upon people. Below a list of potential sources of conflicts is presented with examples in Table 3.2. Although the list is not complete, it covers the potential conflicts which are empirically observed from the services listed in Table 3.1.

3.2.1 Device Conflict

When more than one action is taken on the same device simultaneously, if these actions are *inconsistent* with each other, they have a *device conflict*. For example, Service 17 (S17) from Table 3.1 sends vehicles to the parking garage, where S9 does not allow any outside vehicles into the garage on a special event day. Device conflicts in smart cities can occur on stationary devices (e.g., street lights, traffic signals, message boards, etc.) as well as on mobile devices (e.g., vehicles, drones and robots). Comparing with the stationary devices, conflicts on mobile devices are more difficult to detect and resolve. Without a watchdog layer, the conflict between these two services could not be detected until it really happens. In particular,

- if these actions have opposite directions, but the same numeric parameters, it is an *opposite device conflict*;
- if these actions have different numeric parameters and these parameters cannot be satisfied at the same time, it is a *numeric device conflict*;
- if these actions have the same direction and parameters, but have different durations of application that cannot be satisfied at the same time, it is a *duration device conflict*.

3.2.2 Environmental Conflict

Besides the direct conflicts on shared devices, services are also prone to indirect conflicts caused by unsafe or contrasting effects on the environment (resulting from one or more actions). For example, S6 and S18 from Table 3.1 do not have conflicts on a device because they do not share any device. However, when the air quality is bad, S6 might want to limit the number of vehicles on the road. In contrast, S18 can schedule more buses for passengers at the same time to meet public demand given it is a busy day. As a result, S6 and S18 will cause environmental conflict because of their contrary effects on air pollution level. This is defined as an *environmental conflict*. Environmental conflicts can be categorized into four classes as follows:

- When the set of effects on the environment of a single action causes the state of the city to exceed a safety threshold or violate one/more safety requirements, it results in a *single environmental conflict*.
- When the additive effects on the environment from two or more safe actions exceed the safety thresholds or violate one/more safety requirements, it results in an *additive environmental conflict*.
- When the effects of multiple actions are opposite on the environment and are not approved to happen by the safety or performance requirements, it results in an *opposite environmental conflict*.
- When the effect of one action is the prerequisite of another action, i.e., multiple actions need to be performed sequentially or concurrently, but the previous one is not taken, it results in a *dependent environmental conflict*.

3.2.3 Human Conflict

Humans are at the center of smart city services and cause conflict in several ways. First, smart city services include decision optimization aids, but often also rely on humans working for the city to make final decisions. When 10s or even 100s of humans are making decisions across many service domains, the possibility of conflicts is significant. Because humans are subjective and they tend to make the decision based on personal bias/priority/incentive. For example, in case of co-occurrence of a severe fire and several emergency police calls, decisions on how to dispatch the fire service, police, and ambulances might vary widely among the people who are in charge of these services. Second, human conflicts are caused by contrary effects of environment on human physiology. For example, a decision to temporarily allow greater pollution in an area of the city to ease major traffic congestion can adversely affect asthmatic people in that area. This is a conflict between human physiology and environmental effect. Third, human conflicts can also occur from adverse effects of services on a single person and group.

3.2.4 Typology of Conflicts

While the above list describes under what circumstances conflicts may occur, the below list categorizes various types of conflicts.

Opposite Conflict: An opposite conflict is caused by opposite actions on the same device, on the environment, or on the human taken by different services. For example, traffic congestion service wants to turn the traffic lights to red while the emergency service wants to turn them to green so that an ambulance can pass quickly.

Numeric Conflict: A numeric conflict is caused when actions from different services request different values for one or more parameters of a shared resource. For example, when a storm is coming, the water level monitor service wants to set the water level at X, while the ship management service needs to set the water level to Y to ensure it is high enough for anchoring the incoming ship. Here, conflict will occur if X is unequal to Y.

Duration Conflict: It occurs when two actions are similar and their start time is the same but the duration is different. For example, an accident detection service needs to block the whole road for 20 minutes to deal with an accident, while the traffic congestion service allows the road to be blocked only for 10 minutes.

Completeness Conflict: A completeness conflict refers to the situation when multiple actions are taken by one service to complete a task, but at least one of the actions is affected by the other services, and hence, that service can not complete the task. For example, a water pipe monitor service blocks nearby roads and asks for a crew truck when it detects a serious leak. However, the truck is stuck in a traffic congestion (i.e., effect of traffic congestion service). In this case, the pipe monitor service can not finish the task.

3.2.5 Consequences for Conflicts

Safety Issues: Assume that the smart city sensors detect a traffic accident, or a disturbance such as a fight or riot, or there is a gas or oil leak. These incidents may occur independently or even all at once. Actions taken by smart city safety service upon detecting a unsafe condition might include one or all of the following: dispatch police, ambulance, and/or firefighters, adjust traffic lights to reroute traffic, and inform the public through displays and apps on smart phones. However, other services such as transportation may detect congestion caused by these types of unsafe events and reset the lights or display messages differently than needed by the safety service. For example, the safety service might activate lights to permit an ambulance a non-stop route while the transportation service sets red lights differently to minimize congestion.

In general, it is non-trivial to create a set of services that can predetermine all the possible ways that the services may interact in real-time and under all conditions. This includes conditions such as (i) failure of sensors/actuators, (ii) actions and their consequences are not instantaneous, (iii) occurrence of random events such as a funeral procession or an earthquake, or (iv) entities that are not controllable, e.g., cars not paying attention to the advice/information. This emphasizes the need to create a real-time conflict detection and resolution module that will compare the safety critical nature of conflicts and outcome of various conflict resolution schemes.

Environment and Health Issues: Given environmental monitoring services in smart cities, once pollution is sensed, they will find the causes of the pollution and take actions accordingly. For instance, S6, S7 and S8 can block off the street and limit vehicles, send warning messages to the chemical factories, or prevent ships from coming to the ports of cities. On the other hand, in order to make more profit, commercial services such as the ship management, taxi services and factories may ignore the warnings and even produce more pollutions gradually. Consequently, environment and human health will be impacted negatively. On the other hand, people also cannot close all

the chemical factories and stop all the vehicles to reduce the pollution. Therefore, secondary or even tertiary effects of an action must be addressed while detecting and resolving conflicts. Conflict resolution system also needs to consider the severity of outcomes caused by conflicts, as the outcome can vary widely.

3.3 Requirements in i-CPS

In this section, we present formal specification patterns for a set of typical safety and performance requirements in smart cities. These requirements are originally taken from public documents by U.S. Department of Transportation [90] and U.S. Environmental Protection Agency [91]. Our goal is to provide specification patterns as templates to help people write formal specifications of smart cities more easily. For example, smart city practitioners who are not familiar with temporal logic can instantiate a specification pattern by filling in parameters of the requirement (e.g., locations, time intervals, thresholds). It is expected that as new services are created and deployed there will be a need for additional patterns.

3.3.1 Requirement Formalization

Signal Temporal Logic (STL) [92] is a formalism used to specify real-time properties of discrete and continuous signals. The syntax of an STL formula φ is usually defined as follows,

$$\varphi ::= \mu \mid \neg\varphi \mid \varphi \wedge \varphi \mid \Diamond_{(a,b)}\varphi \mid \Box_{(a,b)}\,\varphi \mid \varphi \mathbf{U}_{(a,b)}\varphi.$$

We call μ a signal predicate, which is a formula in the form of $f(x) > 0$ with a signal variable $x \in \mathcal{X}$ and a function $f : \mathcal{X} \to \mathbb{R}$. The temporal operators $\Box$, $\Diamond$, and $\mathbf{U}$ denote “always", “eventually" and “until", respectively. The bounded interval (a, b) denotes the time interval of temporal operators and can be omitted if the internal is $[0, +\infty)$. Formula $\Box_{(a,b)}\varphi$ is true iff φ is always true in the time interval (a, b). Formula $\Diamond_{(a,b)}\varphi$ is true iff φ is true at sometime between a and b. Formula $\varphi_1 \mathbf{U}_{(a,b)}\varphi_2$ is true iff φ_1 is true until φ_2 becomes true at sometime between a and b.

Next, we present the formal definition of STL quantitative semantics.

$$
\begin{aligned}
\rho(x \sim c, \omega, t) &= \pi_x(\omega)[t] - c \\
\rho(\neg\varphi, \omega, t) &= -\rho(\varphi, \omega, t) \\
\rho(\varphi_1 \wedge \varphi_2, \omega, t) &= \min\{\rho(\varphi_1, \omega, t), \rho(\varphi_2, \omega, t)\} \\
\rho(\Box_I \varphi, \omega, t) &= \min_{t' \in (t, t+I)} \rho(\varphi, \omega, t') \\
\rho(\Diamond_I \varphi, \omega, t) &= \max_{t' \in (t, t+I)} \rho(\varphi, \omega, t') \\
\rho(\varphi_1 \mathcal{U}_I \varphi_2, \omega, t) &= \sup_{t' \in (t+I) \cap \mathbb{T}} (\min\{\rho(\varphi_2, \omega, t'), \\
&\qquad \inf_{t'' \in [t, t']} (\rho(\varphi_1, \omega, t''))\})
\end{aligned}
$$

3.3.2 Safety and Performance Specifications

Table 3.3 shows a set of typical smart city safety and performance requirements in transportation, emergency, and environment services. For each requirement, we provide a specification pattern written in STL. The instantiated specifications can be used to monitor the operation of smart cities at runtime, as shown in Figure 5.1.

The general template is ⟨TL Operator⟩⟨time interval⟩⟨event/action⟩, where ⟨TL Operator⟩ denotes the operator in temporal logic (e.g. Always, Eventually, Until), ⟨time interval is during which time this requirement sustains, ⟨event/action⟩ can be an action or event. In the following, we present templates for creating these specification patterns and describe each specification pattern in detail.

Transportation.

We can use the following templates to generate STL specifications for transportation by filling in parameters of time interval and event:

- Always⟨time interval⟩⟨event⟩, where ⟨event⟩ can be a single objective (e.g., “no collision") or furthered parameterized with the template
  ⟨traffic efficiency metric⟩⟨threshold⟩.

Table 3.3: Smart City Requirements (left) and Specification Patterns (right)

Transportation	
R1: No vehicle collision should occur.	$\square(\neg \mathsf{Collision})$
R2: The number of vehicles in a lane should never exceed its maximum vehicle capacity.	$\square(\mathsf{VehicleNumber(lane)} < \mathsf{MaxCapacity(lane)})$
R3: Traffic congestion in a lane should not increase by more than 10%.	$\square_{(a,b)}(\mathsf{Congestion(lane)} < 1.1 * \mathsf{Congestion'(lane)})$
R4: Traffic yield in a lane should not increase by more than 20%.	$\square_{(a,b)}(\mathsf{Yield(lane)} < 1.2 * \mathsf{Yield'(lane)})$
R5: The average waiting time of vehicles in a lane should not increase by more than 10%.	$\square_{(a,b)}(\mathsf{WaitTime(lane)} < 1.1 * \mathsf{WaitTime'(lane)})$
R6: The number of pedestrians waiting in an intersection by more than 200%.	$\square_{(a,b)}(\mathsf{Pedestrian(i)} < 2 * \mathsf{Pedestrian'(i)})$
Emergency	
R7: Emergency vehicles should not wait for more than 10s at an intersection.	$\square\ (\mathsf{EmergencyWaitTime(i)} < 10)$
R8: Emergency vehicles should not be directed to a blocked lane or area.	$\square\neg(\mathsf{EmergencyDirection(lane)} \wedge \mathsf{Blocked(lane)})$
R9: The highway blocked by an emergency accident should be unblocked within 30 min.	$\mathsf{Blocked(lane)}\ \mathbf{U}_{(1,30)}\ (\neg\mathsf{Blocked(lane)})$
Environment	
R10: The noise level in a lane should always be less than 70 dB.	$\square_{(a,b)}\ (\mathsf{Noise(lane)} < 70)$
R11: The noise level in a lane should be reduced to less than 60 dB after some point.	$\Diamond_{(a,b)}\ (\mathsf{Noise(lane)} < 60)$
R12: The carbon monoxide (CO) emission in a lane should always be no more than 50 mg.	$\square_{(a,b)}\ (\mathsf{CO(lane)} < 50)$
R13: The hydrocarbons (HC) emission in a lane should always be no more than 1 mg.	$\square_{(a,b)}\ (\mathsf{HC(lane)} < 1)$
R14: The particulate matter (PMx) emission in a lane should always be no more than 0.2 mg.	$\square_{(a,b)}\ (\mathsf{PMx(lane)} < 0.2)$
R15: The camera should be turned on and the illumination of street light should be set at least level 3 within a time interval.	$\Diamond_{(a,b)}(\mathsf{Camera(lane)} \wedge \mathsf{Illumination(lane)} > 3)$

- $\mathsf{Eventually}\langle$time interval$\rangle\langle$event$\rangle$, where event can be the number of congested vehicles less than a threshold.

- $\langle$action1$\rangle\mathsf{Until}\langle$time interval$\rangle\langle$action2$\rangle$, where examples of actions include turn a signal light for vehicles/pedestrians to green/red.

R1-*R6* in Table 3.3 are examples of transportation specifications generated using these templates.

R1 is a safety requirement for vehicle collisions. Suppose there is a binary signal Collision, which takes the value True if a vehicle collision occurs. We can, therefore, write a specification $\square(\neg\mathsf{Collision})$ to represent the safety requirement that "No vehicle collision should occur".

R2 is a performance requirement: "The number of vehicles in a lane should never exceed its maximum capacity". The specification uses VehicleNumber(lane) and MaxCapacity(lane) to represent two signals over a location variable lane, which need to be instantiated.

R3 is a performance requirement that compares the traffic congestion state over a time interval (a, b), which is measured by the number of vehicles waiting in a lane. The specification uses signal Congestion(lane) to denote the traffic congestion state in certain lane after the implementation of

some smart services. Congestion′(lane) is a constant obtained from historical data, denoting the average previous congestion state of that lane during that time.

R4 and *R5* are very similar to *R3*, which are all performance requirements measuring the vehicle traffic efficiency. *R4* is about the Yield (i.e., number of vehicles that are unable to cross an intersection where they do not have priority). *R5* is about the average waiting time of vehicles in a lane, denoted as WaitTime(lane).

R6 is a performance requirement about the pedestrian traffic efficiency over a time interval (a, b). The specification uses signal Pedestrian(i) to denote the number of pedestrians waiting at an interaction i after the implementation of some smart services. Pedestrian′(i) is a constant, denoting the average number of pedestrians waiting in an interaction i.

Emergency.

We can use one of the following templates to generate STL specifications for emergency services.

- Always⟨time interval⟩⟨object⟩⟨metric⟩⟨threshold⟩, where the object, for example, can be an ambulance, police car or fire fighter truck, and the metric can be response time and waiting time.
- Eventually⟨time interval⟩⟨action⟩, where the action can be response for an accident, or resolve an accident.
- ⟨action1⟩Until⟨time interval⟩⟨action2⟩, where examples of actions include block a lane, and vehicle/pedestrian moves in a direction/lane.

R7-*R9* in Table 3.3 are examples of emergency specifications generated using such templates.

R7 uses signal EmergencyWaitTime(i) to denote the waiting time of emergency vehicles at an interaction, which should always be less than 10 seconds.

R8 uses two binary signals EmergencyDirection(lane) and Blocked(lane) to represent whether the emergency vehicle is directed to a certain lane and if a lane is blocked, respectively. If both signals are True, then the emergency vehicle is directed to a blocked lane, which violates the safety requirement.

R9 uses Blocked(lane) and ¬Blocked(lane) to denote the block and unblock of a lane. It also uses the Until operator to bound the required time interval.

Environment.

The following template can be used to generate STL specifications for environment requirements:

- Always⟨time interval⟩⟨metric⟩⟨threshold⟩, where metrics can be the level of air pollution and noise.
- Eventually⟨time interval⟩⟨metric⟩⟨threshold⟩, where the time interval can be used to control the time to reduce the pollution level.
- ⟨action1⟩Until⟨time interval⟩⟨action2⟩, where examples of actions include the level of air pollution reaching a threshold or time and allowing vehicles crossing a school area.

R10-R15 are examples of environment specifications generated using these templates.

R10 and *R11* are about the noise level, which use the signal Noise(lane) to denote the noise level in a lane. Similarly, *R12*, *R13* and *R14* use signals CO(lane), HC(lane) and PMx(lane) to denote the carbon monoxide, hydrocarbons and particulate matter emission in a lane, respectively. *R15* uses a signal Camera(lane) to represent that the camera is turned on and a signal Illumination(lane) to denote the illumination levels of street lights. Note that the numerical thresholds used in these requirements are just example parameter values and can be changed.

3.3.3 Smart Services Actions Conflicts

The safety and performance specifications listed in Table 3.3 are all about the states of smart cities (e.g., traffic congestion, air pollution). The actions of smart services can have positive or negative effects on these city states, and thus have the potential of leading to requirement violations and service conflicts. In addition, smart services may issue contradictory actions on the same actuator, which are identified as *device conflicts* in [93]. There are three types of device conflicts. We provide STL specification patterns for each type as follows.

Opposite device conflicts occur when multiple smart services issue opposite (binary) actions to the same actuator. For example, smart traffic service turns a traffic light to green, while smart pedestrian service requests the same light to red. Let $A_1, A_2, \ldots, A_n$ be multiple boolean-valued actions issued

on the same actuator. We can write a specification $\square\neg(A_1 \oplus A_2 \cdots \oplus A_n)$ to detect opposite device conflicts, where $\oplus$ denotes the exclusive or operator.

Numeric device conflicts occur when smart services issue multiple actions with different numeric parameters. For example, the smart energy service tries to set the illumination of street lights to level 1 to save energy, while the smart safety service maintains it at level 3 because the camera to monitor the community requires higher illumination. Let $A_1, A_2, \ldots, A_n$ be multiple real-valued actions issued on the same actuator. We can use the formal specification $\square((A_1 - A_2 = 0) \wedge ... \wedge (A_{n-1} - A_n = 0))$ to monitor if these actions assign the same numeric value.

Duration device conflicts occur when smart services require actions with different time intervals. Let A_i and A_j be two actions. We consider three scenarios. First, A_j cannot be taken within m steps after A_i. Formally, we write $A_i \wedge (\square_{(1,m)}\neg A_j)$. An example is "Traffic light should not be turned to green within 2 seconds after it is turned to red". Second, A_j cannot be taken until A_i stops (e.g., the street should be kept blocked until the accident is resolved). The formal specification is $A_i \mathbf{U}(\neg A_i \wedge A_j)$. Third, we can write $A_i \wedge (\Diamond_{(1,m)} A_j)$ to represent the time dependence between actions. For example, if the accident service blocks an area, it should release blocking eventually within m steps.

3.4 Conflict Analysis Using Real City Data

We first perform an emulation analysis that uses real data from the city of Aarhus, Denmark. Since coordinated services and their related data are not currently available, eight typical services are hypothesized and correlated with real city data. The main purpose of this emulation of services upon real data is to demonstrate the high potential for conflicts. Since many companies and researchers are investigating integration of services across many different domains, it can be expected that in the near future more conflicts will occur.

IoT datasets generated from various sensors in the city of Aarhus, Denmark [94] are used for this analysis. The datasets include vehicle traffic, parking, weather, pollution, cultural events and library events for 61 days (in August and September, 2014). In order to analyze the frequency of conflicts happening across services in a smart city, it is assumed that 8 common services ({S6, S9, S11, S13, S15, S17, S18, S19}) chosen from Table 3.1 are running. These services are chosen because they are either existing policies in cities, or smart applications proposed by researchers in previous papers. By

emulating these 8 services on the Aarhus datasets, the number of times and condition when each service is triggered are determined. Then an assessment of conflicts is done by detecting two or more overlapping service actuations that are contradictory. As a result, the number of conflicts is obtained. Since there is no real data for integrated services this emulation approach is reasonable because (i) the environment analyzed is from real datasets, and (ii) only typical services are used for emulation.

3.4.1 Conflicts between two services

At first potential conflicts between two services issuing requests at the same overlapped time are analyzed. Here, services are assumed to be running separately and the total number of service requests within 61 days is recorded. Then as shown in Table 3.4 9 sets of conflicts were found. For each pair of services in the table, the number of conflicts is shown in column 4. For example, S13 conflicted with S19 32 times. Since these two services are executed a different number of times within 61 days, we also show the percentage of time a service had a conflict with the other service. For example, in row 1 of all the activations of S13, it conflicted with S19 52.5% of its instantiations. Conditions and the corresponding conflicts are also indicated in the table in the last two columns, respectively.

From the results shown in Table 3.4, the following conclusions are drawn:

- Conflicts between 2 services have high frequency, for example, reaching 89% for service 11 conflicting with service 6 (row 9), and being 60.2% on the average for these 8 services.
- The conditions when conflicts occur are very common, but unpredictable.

3.4.2 Conflicts among three services

Conflicts among 3 services are also analyzed. From the conflicts analysis results shown in Table 3.5, the 3-service conflicts also have a high frequency. For example, S11, S17 and S6 conflict almost 70% of the time (calculation is similar to that of Table 3.4). In future smart cities, it is not far-fetched to assume that hundreds of services will be executing concurrently. This data implies that many conflicts are not easily determined ahead of time and must be detected and resolved at run-time.

Table 3.4: Conflicts Analysis between two Services

	S1	S2	Num. of Conflicts	Conflict Prob. with S1	Conflict Prob. with S2	Condition	Conflict
1	13	19	32	52.50%	63.10%	The weather is bad and big events are going on	S13 discourage vehicles driving in bad weather while S19 sends a large number of taxis to the concert after big events
2	15	17	6	34.60%	47.20%	The parking garage near concert is not available and big events are going on in the concert	S15 adjusts traffic lights to reduce/stop vehicles around concert while S17 directs vehicles to the parking garage of the concert
3	9	17	6	34.60%	47.20%	The parking garage near concert is not available and big events are going on in the concert	S17 directs vehicles to the parking garage of the concert while S9 does not allow any vehicles parking without event tickets during events
4	11	17	1795	70.40%	-	The parking garage is not available and heavy traffic congestion around the nearby parking garage	S17 directs vehicles to the parking garage nearby while S11 tries to solve congestion by adjusting traffic signal
5	6	17	2503	98.12%	-	The parking garage is not available and heavy air-pollution around the nearby parking garage	S17 directs vehicles to the parking garage nearby while S6 limits the number of vehicles around to reduce air pollution by adjusting traffic signal
6	6	18	10	55.50%	85.30%	The air quality is bad and big events are just over	S6 limits the number of vehicles around to reduce air pollution by adjusting traffic signal while S18 sends more buses to the concert station after big events because larger flow
7	13	18	32	52.50%	63.10%	The weather is bad and big events are just over	S13 discourage vehicles driving in bad weather while S18 sends more buses to the concert station after big events because larger flow volume
8	6	19	10	55.50%	85.30%	The air quality is bad and big events are about to be over	S6 limits the number of vehicles around to reduce air pollution by adjusting traffic signal while S19 sends a large number of taxis to the concert after big events
9	11	6	650	75.10%	89.20%	Heavy traffic congestion on street i and air quality is bad on the nearby streets	S11 directs vehicles to alternative path to reduce congestion while S6 limits the number of vehicles on that street to reduce the pollution

Table 3.5: Conflicts Analysis among three Services

Services	Conflict Prob.	Condition	Conflict
13,19,18	52.5%	Bad weather after big events	Both buses and taxis are sent to the concert while traffic services limits the number of vehicles on that streets
15,17,9	34.6%	When there is a big event in concert and nearby parking garage is not available	When smart parking sends large number of vehicles to park in rush hour, the streets are blocked off and building does not allow vehicles coming in
11,17,6	69.8%	Parking garage is not available, when streets of nearby garage have heavy air pollution and traffic congestion	When smart parking sends large number of vehicles to park, both traffic services and pollution services are directing vehicles off that streets
19,18,6	55.5%	Heavy air pollution after big events	Both buses and taxis are sent to the concert while pollution services are directing vehicles off that streets
11,13,6	56.1%	Bad weather with heavy air pollution and traffic congestion	Three services have different instructions on the same streets

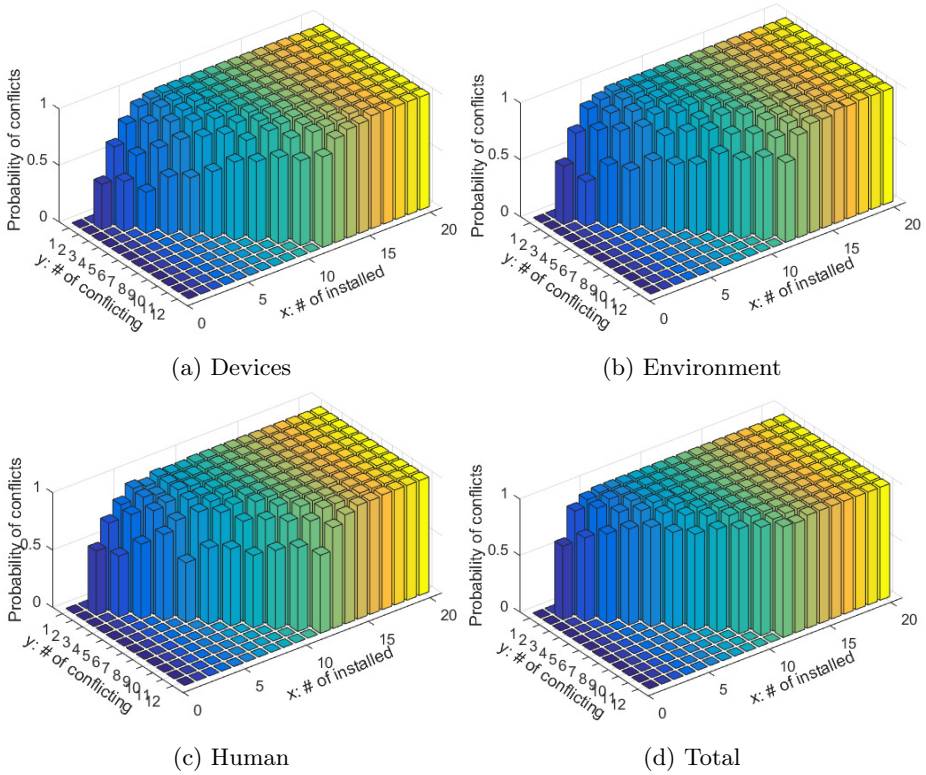


Figure 3.2: Probability of conflicts of future services related to device, environment, human, and the total conflicts of all types. x and y axes represent the numbers of installed and conflicting services, respectively.

3.5 Conflicts among Future Services

A fundamental analysis is conducted that itemizes the conflict frequency among various combinations of these services. The purposes of this analysis are to demonstrate the high degree of potential conflict in the future and to highlight the lessons learned.

There are 41 future services as presented in Table 3.1. The probability of conflicts among them is analyzed. Because a service can send multiple requests to different devices to complete one task at the same time, there can be overlap among the conflicts.

We randomly select n services and compute the average probability of conflicts between them. This process is repeated 100 times. Figure 3.2 shows the probability of conflict between at least m services when n services are running from the 41 services in Table 3.4. With the increasing number of services active, the probability of conflicts grows significantly.

Note the starting point $(2, 2)$ in Figure 3.2, i.e., this point represents the probability of conflicts between at least two services when the two services are running. This analysis is performed between

two services of each type. The device conflicts have the lowest probability, which is only 31%. By contrast, environment conflicts and human conflicts reach 51%, 78%, respectively. Device conflict is lower as when only two services run simultaneously they are less likely to share a device.

Another important point is $(x, 2)$, the number of services running when at least 2 of them have conflicts. This is 10, 7, 12 and 5 for device, environment, human, and total, respectively. This analysis indicates that there is a very high chance that two services have conflicts when as few as 5 services are running in the city. However, there might be hundreds or even thousands of services installed in smart cities in the near future.

The last point in the figure is $(x, 12)$, the number of services running when more than 10 of them have conflicts with others with 100% probability. This result is that 15, 16, 16 and 15 services are running for device, environment, human and total, respectively. Unlike the values of n, values of x are very close. This demonstrates that regardless of the starting probability, probabilities of all types of conflicts increase rapidly with the increasing number of services. The results also indicate that at least 15 services will have conflicts when 20 random services are running.

3.6 Summary

Conflicting services pose serious safety threats and operational failure in a smart city environment. This chapter focuses on formulating the problem of conflicts and showing that it is a serious problem. Specifically, it (i) outlines several characteristics of services that contribute towards conflicts, (ii) proposes a conflict taxonomy in terms of origin of conflict, (iii) lists 41 potential services across five domains (transportation, safety, environment, emergency and energy) for a smart city, and (iv) outlines issues and research challenges of detection and resolution of conflicts. Our evaluations using real data and services demonstrate the high probability of conflicts in smart cities. It shows the significance of conflicts in i-CPS.

Chapter 4

Conflict Detection and Resolution System

Advances in technologies such as the Internet of Things have transformed the way cities operate. For example, sensors and actuators on streetlights are installed and used to gather information about traffic, street parking and pollution, to adjust LED streetlights to save energy, and to estimate the size of crowds in responding to public disturbances. Various smart services are built to improve the performance and efficiency of smart cities across different domains (e.g., transportation, energy, healthcare). It is estimated that a smart city can generate revenue and cost savings of $2.3 trillion globally through 2024 [95].

In the previous chapter, we identified different types of conflicts among smart services and their severe safety consequences. However, most of existing solutions [16, 18] adopt a simple approach to conflict resolution by only accepting actions with the highest priority and rejecting others, which does not account for the adaptive policies of action priorities and optimal resolution for city requirements.

Nevertheless, detection and resolution of conflicts across services in smart cities is very complex. The requirements objectives in smart cities are often expressed vaguely in natural language and are potentially conflicting. For example, a resolution that satisfies the requirement of reducing traffic congestion may actually violate another requirement of maintaining noise levels. There is often no

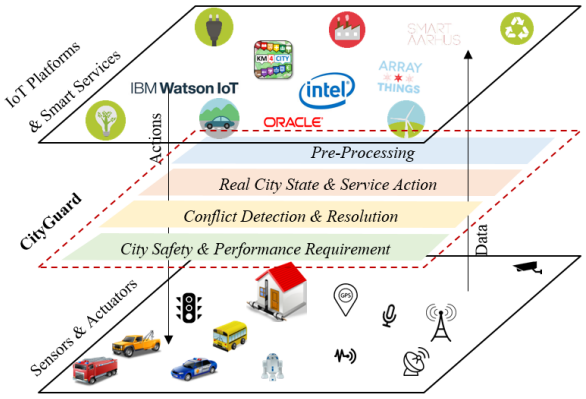


Figure 4.1: Conflict Detection and Resolution in the Smart City

best resolution and/or one cannot satisfy all objectives, which implies that trade-offs must occur. In addition, priorities of smart services and actions may be adaptive to the current state of the city. For example, resolving conflicts of traffic congestion may be more urgent at rush hours than at other times. Moreover, there are uncertainties in the state of the city, in human behaviors, and the impact of the resolutions (in time and space). The scale of smart cities and services also makes it challenging to search for an optimal resolution from an enormous solution space.

In this chapter, we present a decision support system for conflict detection resolution in smart cities. The system includes two important components, a conflict detection component (i.e., CityGuard) and a conflict resolution component (i.e., CityResolver). First, CityGuard detects different types of conflicts by intercepting actions ahead of time, analyzing the details of the actions, and then running simulations to predict potential conflicts within a temporal and a spatial range. This chapter focuses on conflict detection, especially for environmental conflicts. Next, CityResolver uses a novel Integer Linear Programming (ILP) based method to generate a small set of candidate resolution options. The ILP-based method makes it more efficient to search for optimal resolutions from an exponentially growing number of possible resolution choices, and it considers adaptive policies of service priorities that change as a function of context. It then checks these resolution options' impact on city performance using a Signal Temporal Logic (STL) based verification approach. Formalizing city requirements expressed in natural language using STL specification helps to resolve the vagueness of requirement objectives. The verification is performed on predicted traces of future city states, which are generated from the simulation of executing resolution options in the smart city. The simulation also accounts for uncertainties and disturbances in the city (e.g., weather and scheduled events). Given a resolution option, the STL-based approach verifies if the predicted city states violate

city requirements. It computes the degree of requirement violations based on three different metrics: (1) the robustness value, (2) the percentage of time when violations occur, and (3) the integral of signal deviations. Our system provides a trade-off analysis of different resolution options' effects on various city requirements. The results are plotted on a dashboard to support decision makers to choose the best resolution. We apply a prototype implementation of our system to a case study of a simulated smart city based on the map of lower Manhattan, New York. We demonstrate the effectiveness of our system by comparing the performance with two baselines: a smart city without conflict resolution, and CityGuard which uses a priority rule based conflict resolution. Experimental results show that our system reduces the number of requirement violations and improves the city performance significantly.

4.1 Motivating Example

In this section, we describe an example of smart services and their conflicts, based on a map of the lower Manhattan district in the New York City (Figure 4.2). We assume that there are 10 different smart services in this district, which are overseen by a smart city operations center. We only introduce four services here, and list all ten services in Table 4.6. **S1** is a smart traffic service, which can control traffic signals in street intersections to relieve congestion and optimize or improve traffic performance. **S2** is a smart emergency service, which can request green traffic signals in order to transport patients in critical conditions to hospitals as soon as possible. **S3** is a smart accident service, which can block a street where some accident occurs and alert nearby vehicles to detour. **S4** is a smart infrastructure service, which can schedule infrastructure check-up and repair appointments. The operations of these smart services have to satisfy a set of safety and performance requirements in the smart city. For example, **R1** is an environment requirement that the noise level in the school area should always be less than 50db. **R2** requires that the Carbon Monoxide (CO) emission in an intersection should always be less than 40mg. **R3** is a requirement for transportation domain, which requires that the waiting time of the traffic in an intersection should not be greater than certain threshold λ. **R4** specifies that an emergency vehicle should not wait in an intersection for more than 10 seconds. A complete list of requirements considered in this paper can be found in Table 4.2.

Conflicts may arise when different smart services request contradicting actions on the same actuators at the same time, or when the effects of service actions violate the smart city's safety and performance requirements. For example, suppose that S1 requests longer green traffic signals on 1st Avenue to

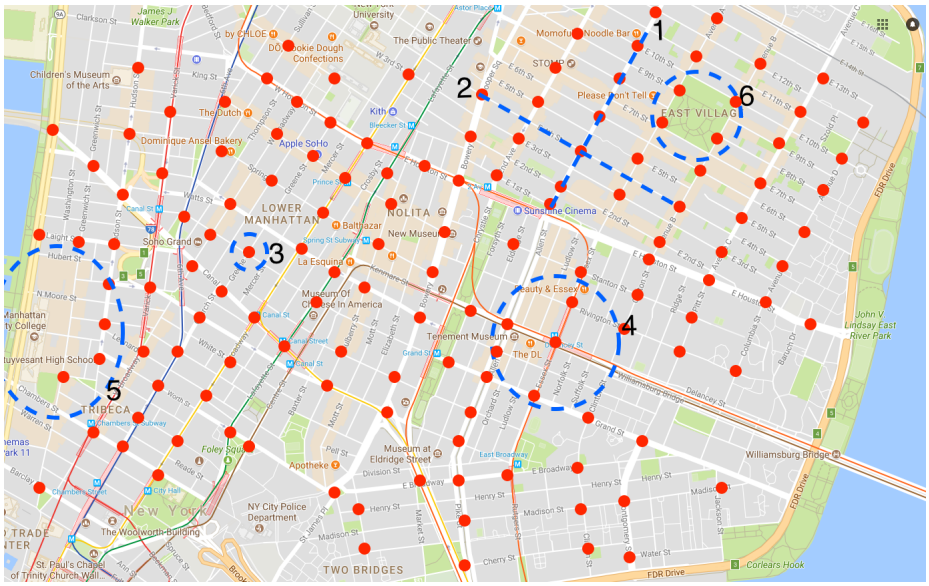


Figure 4.2: A map of the lower Manhattan district in New York City. Red dots: street intersections with hypothetical smart sensors and services. Blue dashed lines/circles: areas of interest (1: 1st Avenue, 2: East 4th Street, 3: The intersection of Greene Street and Grand Street, 4: Blocks around the entry of Williamsburg Bridge, 5: Stuyvesant High School and Borough of Manhattan Community College, 6: East Village).

relieve traffic congestion, while S2 requests green traffic signals on East 4th Street to transport a patient in an emergency. The requested actions by S1 and S2 contradict each other on the traffic signal at the interaction of 1st Avenue and East 4th Street (annotated as areas 1 and 2 in Figure 4.2). In addition, the effects of actions requested by S1 and S2 may cause an increased traffic in nearby East Village (area 6 in Figure 4.2), and thus violating requirements about noise levels (R1) and CO emission (R2).

The smart city operations center detects such conflicts and provides resolutions by accepting or rejecting smart services' action requests. However, it is very challenging to find an optimal or even acceptable conflict resolution. First, the feasible set of resolutions grows exponentially with the increasing number of smart services and actions. An exhaustive search over the entire solution space is not efficient or even possible. Second, what does it mean by optimal when considering multiple smart city requirements that are expressed vaguely in English? For example, one resolution may dramatically reduce traffic congestion, but increase pollution levels. An ideal resolution should balance the trade-off between multiple objectives. Third, the severity of conflicts and the importance of service actions are often dependent on the current state of the city. For example, during rush hour, resolving traffic congestion is more urgent than maintaining noise levels and CO emissions. Thus, the conflict resolution should consider an adaptive policy of prioritizing smart services and

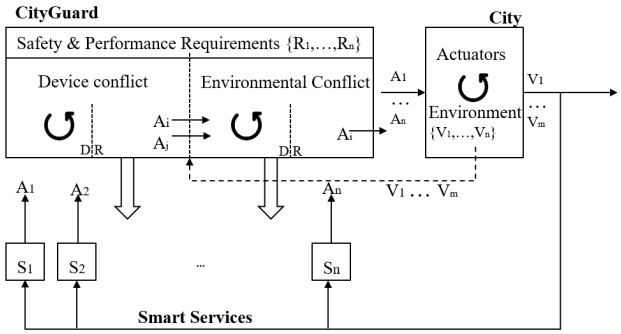


Figure 4.3: CityGuard Running in a Smart City

requirements. Finally, there are many uncertainties in the state of the city, including disturbances that are predictable (e.g., weather, events) and unpredictable events (e.g., accidents). To address these challenges, we propose a decision support system for conflict resolution in smart cities as described in the next section.

4.2 Conflict Detection

CityGuard is a safety watchdog built between the smart services and the infrastructure layers (see Figure 1). CityGuard executes as a feedback loop as shown in Figure 4.3. Variables V_1 through V_m monitor city states, and services S_1 through S_n use the monitored data to choose actions. CityGuard intercepts the actions, decides if there is a device, environment, or both conflicts based on the safety and performance requirements. Unsafe actions and conflicts are detected and resolved through CityGuard. Safe actions are taken in the city and cause the change of city states, which triggers actions from smart services again. As a result, the goal is that only safe actions are executed in the city. However, CityGuard does not guarantee safety, but rather significantly improve it as shown in the evaluation.

The internal structure of CityGuard is shown in Figure 4.4. There are 4 important components in CityGuard, *City Safety and Performance Requirements (CSPR), City State and Service Action (CSSA), Pre-processing, and Conflict Detection and Resolution (CDR).* Algorithm 1 shows how the components are executed.

4.2.1 Safety and Performance Requirements Component

The safety and performance requirements component provides the rules for CityGuard to monitor all the actions. It has three modules: (i) principles, (ii) requirements, and (iii) updating requirements.

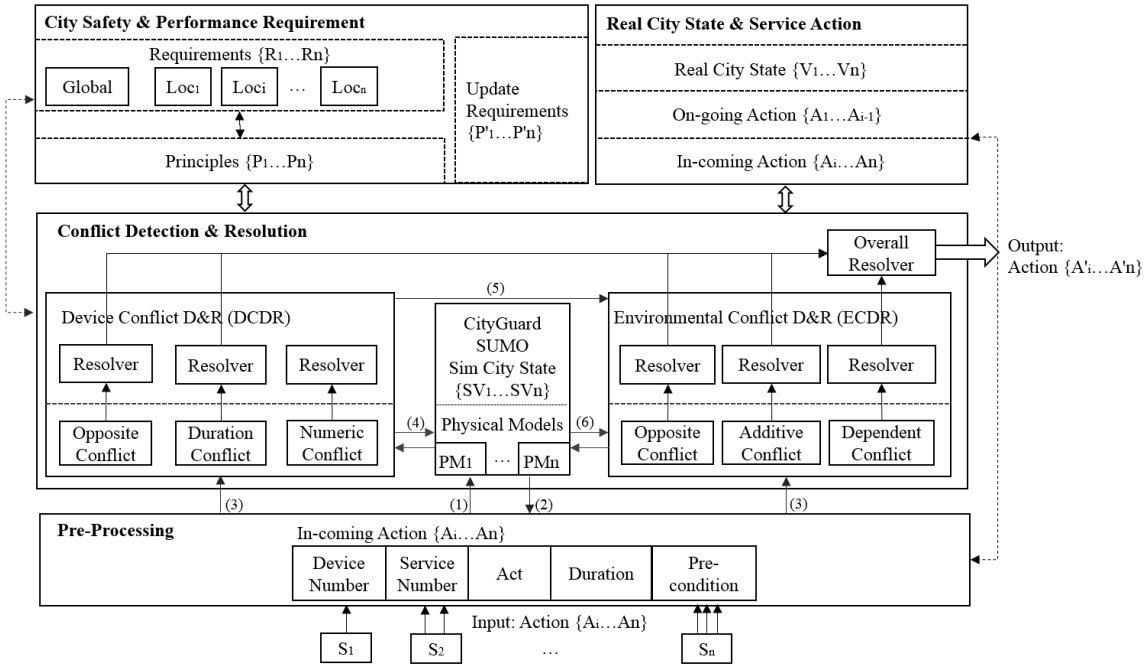


Figure 4.4: CityGuard Structure (*Pre-Processing* intercepts and checks the safety of the single action, *Conflict Detection & Resolution* detects conflicts among actions after Pre-processing, and *City Safety & Performance* and *Real City & Service Action* store the safety requirements and city states, respectively. Section III describes each component in detail.)

All principles and requirements are defined and specified by city personnel. CityGuard integrates them into the safety checking components.

To start with, CityGuard follows a set of principles to maintain safety requirements. For example, it might contain,

- Any action of a service should not violate predefined city / individual service safety requirements.
- A safety requirement is a function of location and time with conditions.
- When there is a conflict between different safety requirements, follow the city objectives.

CityGuard works with the safety and performance requirements specified by a particular smart city that is running CityGuard. For example, important safety and performance requirements for a smart city might include (i) Noise levels should be below the following thresholds, community/school (Day 50 db, night 45 db), Mall/working zones (Day 60 db, night 50 db), Highway (Day 70 db, night 55 db); (ii) Actions taken by transportation services should not cause collision of vehicles ; and (iii) emergency vehicles should not wait for more than 10seconds at any intersection.

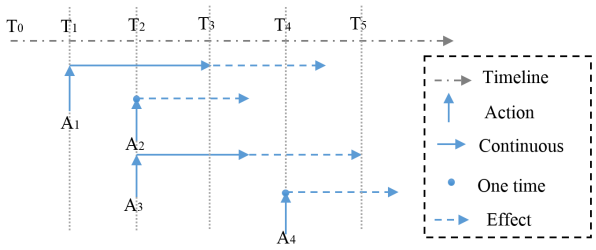


Figure 4.5: Effects of Actions Taken in the Smart City Over Time

These requirements, defined in English by the city, are integrated and translated to formal metrics in CityGuard manually. For example, the above requirements can be translated to (i) R_1: $Noise(Location, Time) < xdb$, (ii) R_2: $Num(collision) < 0$ and (iii) R_3: $waitingTime(E) < 10s$.

Since the mapping between actions and safety requirements is not always straightforward, CityGuard simulates and analyzes the effects of an action on the metrics and therefore decides if it is a safe action by examining if the metrics are within their requirements.

Furthermore, with new services added and situations changed in the city, safety and performance requirements are also added and updated.

4.2.2 Real City State and Service Action Component

City state and service action is considered as the interface of CityGuard to services and cities. It obtains real city states and passes them to CityGuard SUMO in the CDR, obtaining a consistent view of states in the real city and the simulated city. Meanwhile, it also stores and provides the in-coming actions from services and on-going actions in cities.

4.2.3 Pre-Processing Component

When an action A_i is intercepted by CityGuard, it contains information to direct an actuator, which is also the source for CityGuard to analyze potential conflicts. Usually, an action has the following information, *Device Number* indicates a unique numerical identifier of the actuator on which the action is supposed to be taken. *Service Number* indicates a unique numerical identifier of the service that issues the action. *Act* is the expected action or effect, which depends on the functions of the services and could be a change in states, locations, or send a warning or message, etc. *Duration* is the requested duration of this action. Some actions are continuous actions and need to be acted

Algorithm 4.1: CityGuard

input : ActionSet$\{A_k\}$
output : SafeActionSet$\{A'_k\}$
initialize : CityState$\{V_k\}$, SimulationState$\{SV_k\}$, Requirement$\{R_k\}$, SimuStep = 0
while $\{A_k\}$! = 0 **do**
  Pre-Processing:
  for action = 1 : length($\{A_k\}$) **do**
    $\{SV_k\}$ = $\{V_k\}$;
    for SimuStep = 1 : n **do**
      $\{SV'_k\}$ = CityGuardSUMO(A_k, $\{SV_k\}$);
      f_{FC} = SafeCheck($\{SV'_k\}$, $\{R_k\}$);
    end
    if f_{FC} == 1 **then**
      A'_k = CityResolver(A_k);
    end
  end
end
DeviceConflict:
f_{DC} = DeviceCheck ($\{A'_k\}$);
if ($f_{DC_{ij}}$ == 0)|($f_{DC_{ij}}$ == 1&&A_i == A_j&&$Dur(A_i)$ == $Dur(A_j)$) **then**
  go to EnvironmentConflict
end
if $f_{DC_{ij}}$ == 1 **then**
  if A_i == $-A_j$ **then**
    A'_k = CityResolver(A_i, A_j)
  end
  if $A_i + A_j > R_k$ **then**
    A'_k = CityResolver(A_i, A_j)
  end
  if $Dur(A_i)$! = $Dur(A_i)$ **then**
    A'_k = (CityResolver(A_i, A_j))
  end
end
EnvironmentConflict:
$\{SV_k\}$ = $\{V_k\}$;
for *SimuStep = 1 : n* **do**
  $\{SV'_k\}$ = CityGuardSUMO(A'_k, $\{SV_k\}$);
  f_{EC} = check ($\{SV'_k\}$, $\{R_k\}$);
  if f_{EC} == 1 **then**
    A'_k = CityResolver(A_i);
  end
  if $Ef(A_i)$ == $-Ef(A_j)$ **then**
    A'_k = CityResolver(A_i, A_j)
  end
  if $A_i \leftarrow A_j$&& *exist(A_j) == 0* **then**
    A'_k = (CityResolver(A_i))
  end
end
$\{A'_k\}$ = CityResolver($\{A'_k\}$);

on for a certain time, while some actions are just one time actions. *Pre-conditions* indicate the pre-conditions of the action, mainly pointing to the essential concurrent or sequential actions. The format is <ActionID, Con/Pre>.

In a pre-processing phase, CityGuard checks the above action information and deals with any missing information. Device number and service number are easy to tell from the source and destination of the action. All *acts* are assumed to be contained in the action information, otherwise, it cannot be executed by the actuator. Unless specified, the duration is treated as 0 if it is missing on the action

information. It is a reasonable way to deal with missing duration because even if a continuous action is being treated as a one-time action, it still has an effect on the city performance, which will be detected by the environmental conflict detection procedure.

After intercepting the actions, CityGuard also needs to consider the timing of the actions. CityGuard defines three types of actions based on their action and effect times, including In-coming actions, On-going actions, and Past actions. An In-coming action is one just being intercepted and going to be checked by CityGuard. An on-going action is the one that has been already checked by CityGuard and is running. Because it is still using the device, the in-coming action which wants to take different action on that device may be conflicting with it. A past action is the one that has been taken and finished. Though it may still have an effect on the city, its effects are reflected by the city environmental states. Therefore, CityGuard does not track these actions any more. For example, in Figure 4.5, at T_2, A_1 is an on-going action, A_2 and A_3 are in-coming actions. Thereby, all of A_1, A_2 and A_3 may be conflicting and need to be checked by CityGuard. However, at T_4, when A_4 comes in, there is no on-going action, then CityGuard only needs to check if T_4 is a safe action in terms of the single action environmental conflict.

As a result, three key parameters are retained by CityGuard, i.e., In-coming actions $\{A_i, ..., A_n\}$, On-going actions, and the City States $\{V_1, ..., V_{i-1}\}$.

Three steps are performed in the pre-processing,

Step 1: Intercept actions and obtain their key information.

Step 2: Check single actions' safety by running them in CityGuard SUMO. Send back unsafe actions with warnings to their services. ((1)(2) in Figure 4.4)

Step 3: Check Device Number of in-coming and on-going actions, then send the actions with same device number to DCDR because they have a potential device conflict. Pass the other actions to the ECDR. ((3) in Figure 4.4)

After pre-processing, all the actions sent to the conflict detection components are safe single actions.

4.2.4 Conflict Detection Component

Conflict Detection Component consists of 4 sub components, i.e., CityGuard SUMO, Device Conflict Detection and Resolution (DCDR), Environmental Conflict Detection and Resolution (ECDR), and an Overall Resolver (OR). In this section, we first focus on the key solutions for conflict detection. In the next section, we will show more details of a sophisticated conflict resolution approach (i.e., CityResolver).

CityGuard SUMO is the central component of the CDR which is used by both DCDR and ECDR to simulate the effects of actions on a real city. The solution uses the Simulation of Urban MObility (SUMO) [11], a traffic simulation that models inter-modal traffic systems including road vehicles, public transport, and pedestrians. By implementing smart services and simulating real city scenarios in SUMO, CityGuard SUMO plays an important role to test the primary and secondary effects of actions. To do this, there are different Physical Models (PM) in CityGuard SUMO to simulate the primary and secondary effects of actions. For example, the traffic - air PM knows how the numbers and speeds of different types of vehicles affect emissions (e.g., CO, HC, PM) quantitatively. As a result, the secondary effects on environments of actions from transportation services are obtained.

Before executing, CityGuard inputs into the simulator the same states of the city where actions are going to be taken. Then, it runs one or multiple actions in this scenario into a future time interval. New states of the city after taking these actions are sent back to DCDR and ECDR, where the decisions of detection and resolution conflicts are made. SUMO also has models to simulate accidents.

Device Conflict Detection and Resolution Component

This component is shown in the left part of CDR in Figure 4.4. Three types of device conflicts are processed using different detectors and resolvers with corresponding models. Once potential device conflict actions are received by DCDR, their *acts* are sequentially checked by opposite, duration, and numeric conflict detection modules with the steps described below.

Step 1: Following the logic defined in Section II, DCDR compares the given actions to detect if they are (i) opposite, (ii) with different numeric requests, or (iii) the same. Accordingly, proceed to Step 2 through Step 4.

Step 2: If they are (i) opposite, call the opposite conflict resolver, which makes a decision according to ORR.

Step 3: If they are (ii) with different numeric requests, whether they are conflicting depends on if they can be taken at the same time, which is simulated in CityGuard SUMO. If one action with the larger numerical request actually tolerates the others, there is no numeric conflict because all of the actions can be satisfied. Otherwise, the numeric resolver is called for decision making according to ORR.

Step 4: If they are (iii) the same, compare their *durations*. If durations are the same, there is no device conflict between/among them and the actions are sent to ECDR; If durations are different, similar to Step 3, actions are simulated, and tolerance is checked. A longer duration is accepted by its resolver if they are tolerant. Otherwise, a decision is made according to ORR.

Step 5: All actions with a marked decision from DCDR are sent to ECDR.

Environmental Conflict Detection and Resolution Component

All the actions received by this component (shown on the right side of CDR in Figure 4.4) from the Pre-processing and DCDR components are sent to run in CityGuard SUMO for N steps to see their combined effects on the environment (see (6) in Figure 4.4). Through analyzing performance results from CityGuard SUMO and checking with the safety and performance requirements, ECDR detects three types of environmental conflicts with following steps.

Step 1: Check if their additive effects conflict with the city requirements, which includes the situations when simulated states exceed required thresholds and when forbidden unsafe cases happen. If so, then there is an additive conflict. The additive resolver is called.

Step 2: With the results from CityGuard SUMO, the environmental opposite conflicts are detected by comparing the primary effects detected from pre-processing component with the combined effects from ECDR. If there is an opposite conflict detected, actions are sent to the corresponding resolvers. If these opposite effects on the environment violates safety requirements, a resolver resolves it according to ORR. Otherwise, the resolver decides whether to resolve it or not based on the context of the system.

Step 3: In the dependent conflict detection model, pre-conditions of actions are checked.

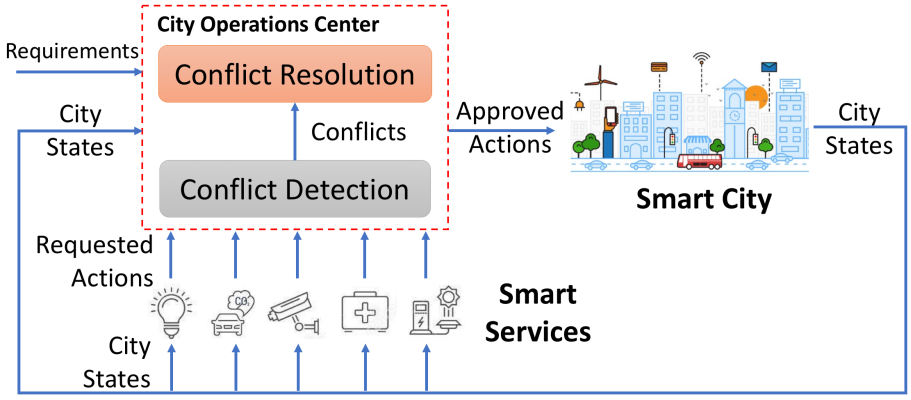


Figure 4.6: Overview of conflict detection and resolution among smart services in smart cities.

Step 4: All marked actions are sent to the overall resolver.

Resolvers in the DCDR and ECDR and Overall Resolution Rules (ORR) for decision making are presented in the following section.

4.3 Conflict Resolution

We envision a watchdog architecture (e.g., CityGuard [93]) in which a city operations center would oversee all smart services, detect conflicts among service requests, and provide resolutions. Such a city operations center could follow real-world prototypes including IBM's Rio de Janeiro Operations Center [96] and Cisco's Smart+Connected Operations Center [97], where real-time information about city states (e.g., traffic, pollution) are collected from citywide sensors and displayed on the command room's monitors. Figure 4.6 shows an overview of our envisioned architecture that extends the functionality of a city operations center with conflict detection and resolution. Smart services send action requests based on real-time city states. The city operations center intercepts these action requests and detects if there is any conflict that would lead to contradicting actions or violations of city requirements. If no conflict is detected, all action requests are approved for execution in the smart city. Otherwise, an optimal resolution is computed to resolve the detected conflicts. We refer readers to our previous papers [93, 98] for conflict detection methods, and focus on addressing challenges of conflict resolution.

We present CityResolver – a decision support system for conflict resolution in smart cities. An overview of CityResolver is shown in Figure 4.7. Suppose one or more conflicts between smart services' action requests are detected.

The first step is to generate a set of resolution options, each of which may accept a subset of action requests and reject the others. The resolution options may also suggest alternatives or delayed executions of requested actions. Thus, the number of potential resolution options grows exponentially with the number of action requests. We develop an Integer Linear Programming (ILP) based method to select a small set of candidate options, accounting for policies that define the priorities of the smart services and their actions. These policies are not fixed, but are adaptive based on current city states (context). We will describe the ILP-based method in Section 4.3.1.

The second step is to simulate the execution of these resolution options in order to predict the effect of choices on the city. Here, we use an off-the-shelf city simulator [99]. Multiple simulations may be instantiated in parallel to simulate the execution of smart city under different resolution options. For each resolution option, the simulator starts with the city states at the current time t (i.e., when conflicts are detected) and simulates the city executing the resolution option for a period of Δ_t into the future. Traces (time series) of city states from t to $t + \Delta_t$ are generated for verification that an option does not violate safety and performance requirements. The simulation also accounts for disturbances in the city (e.g., heavier traffic during 5 to 7 pm, a 80% chance of rainy day, or a big event is scheduled). We distinguish two types of disturbances or uncertainties in smart cities: predictable and unpredictable. The simulation only considers predictable disturbances. However, unpredictable disturbances (e.g., accidents, device failures) are handled in CityResolver due to a continuous feedback loop that is monitoring city states in real-time (see Figure 4.6). If these states change greater than associated set points then the services themselves issue new actions which we intercept and re-apply the detection and resolution actions.

The next step is to verify if the simulated traces of city states of each option satisfy various city requirements. We develop an approach to formalize smart city requirements as Signal Temporal Logic (STL) specifications and compute the trade-off between different resolution options on multiple specification objectives via STL verification (see Section 4.3.2).

The trade-offs between options are displayed in a decision support dashboard. The decision maker chooses a resolution, by comparing the performance of different options on various city requirements. For example, Figure 4.8 shows the trade-off between three options on requirements R1-R4. We will describe these options and their generation later in Example 4.1 in Section 4.3.1. The values in the trade-off display represent the violation degree in terms of the percentage of time when the requirement is violated. The zero value means that the requirement is not violated. Figure 4.8 shows

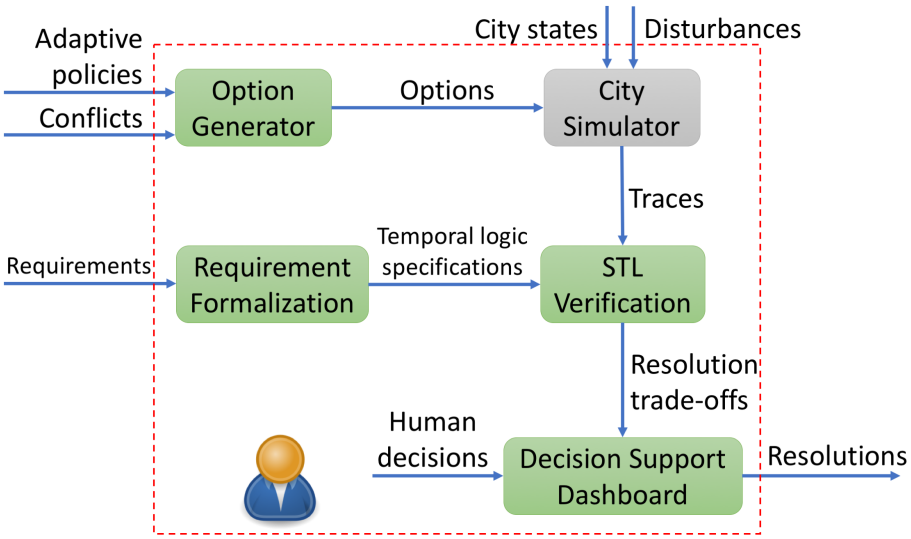


Figure 4.7: Overview of CityResolver – a decision support system for conflict resolution in smart cities.

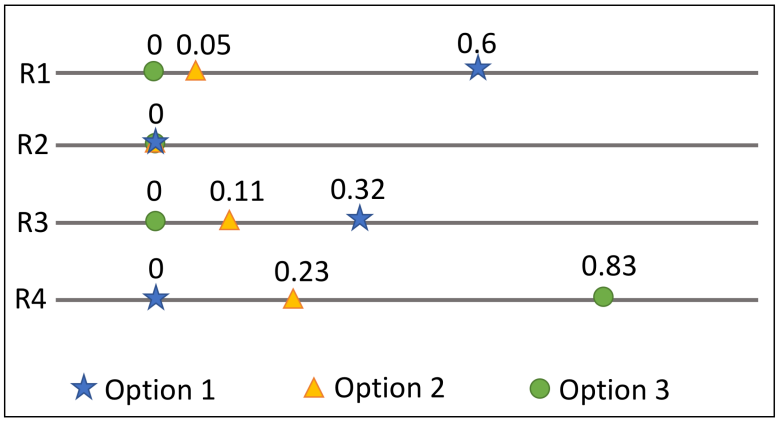


Figure 4.8: An example dashboard displaying the trade-off between three resolution options in terms of the percentage of time violating R1-R4. (Example 4.1 describes these options.)

that Option 3 satisfies the first three requirements but violates R4 most of the time, while Option 1 satisfies R2 and R4, but violates R1 and R3. The human decision maker may choose a resolution option based on individual preferences. To reduce the human burden of selecting resolutions every time a conflict occurs, CityResolver also allows the automatic selection of optimal resolutions based on a set of rules predefined by the human decision makers. Suppose that a human decision maker thinks that R4 is more important than other requirements and defines a rule that the optimal resolution should not violate R4. Then Option 1 is automatically selected based on the rule.

4.3.1 Generating Resolution Options

In this section, we present an Integer Linear Programming (ILP) based method to generate a small set of resolution options, which corresponds to the "Option Generator" module in Figure 4.7.

Suppose that there are m on-going smart service actions executing in the smart city without any conflict. The city operations center intercepts n new action requests from smart services and detects that there are some conflict between these $m + n$ actions. One strategy to obtain resolutions is to only accept some of the new actions while rejecting others in a way that there are no conflicts. To achieve this it may also be necessary to suspend some on-going actions. Thus, the number of possible resolution choices is at least 2^{m+n}. If we consider a more complex resolution strategy, such as suggesting alternative actions to the requested actions or putting the rejected actions in a waiting list for delayed execution, then the solution space of possible resolutions becomes even larger. It would be very challenging, if not impossible, to check all these resolution choices' impact on city states and requirements within the short time frame of resolution decision making. Thus, we present a method to select a small number of candidate resolution options based on the intuition that a good resolution should (1) accept as many actions as possible, (2) not allow contradicting actions, and (3) account for priorities of services and actions.

We formulate the problem as an integer linear program. Given a set A of smart service actions causing conflicts, we define a binary variable $\mu \in \{0, 1\}$ for each action $a \in A$ to track if the action is chosen by a candidate resolution option. Each action a is associated with a weight value $w \in \mathbb{Z}$, representing the action priority determined by current, state-dependent importance policies. For simplicity, we assume that action weights are given as constants at time t. We denote a set C of contradicting action pairs and a set D of dependent action pairs. We also group an action and its alternatives into a set $\theta \subseteq A$. The resulting ILP problem is

$$\underset{w_i \in \mathbb{Z},\ \mu_i \in \{0,1\}}{\text{maximize}} \sum_{1 \le i \le |A|} w_i \times \mu_i \tag{4.1}$$

subject to

$$\forall (a_i, a_j) \in C: \ \mu_i + \mu_j \le 1, \tag{4.2}$$

$$\forall (a_i, a_j) \in D: \ \mu_i - \mu_j = 0, \tag{4.3}$$

$$\forall a_i \in \theta \subseteq A: \ \sum \mu_i \le 1 \tag{4.4}$$

The objective function (4.1) is to maximize the number of accepting actions in the resolution based on their priority weights. The constraint (4.2) guarantees a resolution does not accept a pair of contracting actions. The constraint (4.3) ensures that dependent actions are both accepted or rejected at the same time. Finally, the constraint (4.4) requires that at most one action from a set of alternative actions is chosen by a resolution. Transforming the problem to ILP and solving it with the Gurobi tool do not necessarily find the best solution when the number is very large, but it can give the solution in polynomial time, which is very important for runtime decision making system in cites.

We illustrate the usage of the ILP solution below.

Example 4.1. *Suppose that smart traffic service S1 requests seven traffic signals on the 1st Avenue to stay green for 5 minutes. The requested actions are denoted as* $\{a_1 ... a_7\}$*, corresponding to traffic signals drawn as seven red dots (from south to north) on street 1 in Figure 4.2. If action* a_3 *is not accepted, the service also allows an alternative action (denoted by* a_8*) to keep the corresponding signal green only for 3 minutes. Suppose that, at the same time, the smart emergency service S2 requests three green traffic signals on the East 4th Street for 3 minutes. The actions are denoted as* $\{a_9, a_{10}, a_{11}\}$*, corresponding to traffic signals drawn as three red dots (from west to east) in street 2 in Figure 4.2. Actions* a_9 *and* a_{10} *are interdependent. Action* a_{10} *is contradicting with actions* a_3 *and* a_8*. Actions requested by the emergency service S2 has a higher priority weight than the traffic service S1. Let the weight value for S2 actions be 2 and the weight value for S1 actions be 1. We write an ILP as follows:*

$$\underset{\mu_i \epsilon \{0,1\}}{\text{maximize}} \sum_{1 \le i \le 8} \mu_i + \sum_{9 \le i \le 11} 2 \times \mu_i$$

subject to

$$\mu_3 + \mu_{10} \le 1$$

$$\mu_8 + \mu_{10} \le 1$$

$$\mu_9 - \mu_{10} = 0$$

$$\mu_3 + \mu_8 \le 1$$

We rank solution results based on their objective function values. The top 3 resolution options are as follows.

- *Option 1: Reject a_3 and a_8, accept other actions.*
- *Option 2: Reject a_3, a_8 and a_{11}, accept other actions.*
- *Option 3: Reject a_8, a_9 and a_{10}, accept other actions.*

A trade-off between these options is shown in Figure 4.8.

4.3.2 Verifying Resolution Options

In this section, we describe how to compute the trade-off between different resolution options via Signal Temporal Logic (STL) based runtime verification of city requirements.

Requirement Formalization

In Chapter 3, we provide a set of templates for expressing typically safety and performance requirements in smart cities as STL specifications. We find that most smart city requirements can be specified using STL formula in the form of $\Box_{(a,b)}\ (x < \lambda)$ where x is a signal about city state and λ is a threshold.

Example 4.2. *We translate requirements R1-R4 (described in Section 4.1) to STL formulas as follows.*

- *STL formula for R1:* $\Box_{(0,\Delta t)}\ (\mathsf{Noise} < 50)$.
- *STL formula for R2:* $\Box_{(0,\Delta t)}\ (\mathsf{CO} < 40)$.
- *STL formula for R3:* $\Box_{(0,\Delta t)}\ (\mathsf{WaitTime} < \lambda)$.
- *STL formula for R4:* $\Box_{(0,\Delta t)}\ (\mathsf{EmergencyTime} < 10)$.

Computing Requirement Violation Degrees

We now describe how to compute the degree in which a continuous signal about smart city state violates a city requirement $\Box_{(a,b)}\ (x < \lambda)$. A notion of robustness value for satisfying or violating STL formulas is formally defined in [92]. The robustness value of a given signal x violating $\Box_{(a,b)}\ (x < \lambda)$

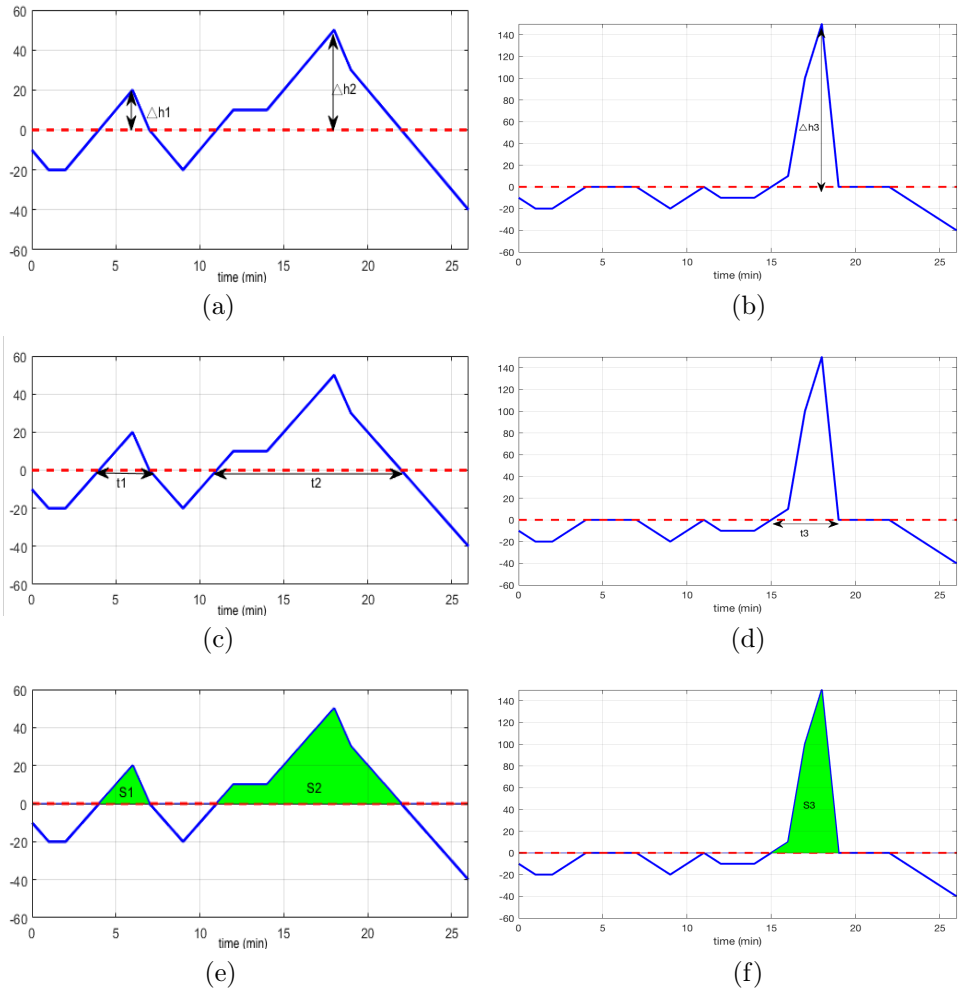


Figure 4.9: An example of two options showing three metrics of violation degrees for STL formula $\square_{(0,25)}$ (Noise < 50). Blue solid curves represent the signal (Noise − 50) under resolution option (i) and (ii). For option (i), (a) the robustness value: Δh_2, (c) the percentage of violation time: $(t_1 + t_2)/25$, and (e) the integral of deviation: $S_1 + S_2$; For option (ii), (b) the robustness value: Δh_3, (d) the percentage of violation time: $t_3/25$, and (f) the integral of deviation: S_3.

at time τ is defined as

$$\rho = \sup_{t\epsilon(\tau+a,\tau+b)}(x(t) - \lambda). \tag{4.5}$$

Intuitively, the robustness value indicates extremum points of the signal. The robustness value is useful for telling the worst-case performance, but it does not show the average or overall performance. For smart cities requirements, we are interested to know both. We use the following example to illustrate why measuring the robustness value only is not sufficient for finding optimal resolutions.

Example 4.3. *Suppose that the noise level is between 50 db and 100 db for 12 min with option (i), and over 100 db for 2 min with option (ii). We verify the city state against requirement*

$\square_{(0,25)}$ (Noise < 50). *Figure 4.9(a) plots the signal of* (Noise − 50) *under option (i) and indicates that the robustness value of violating the requirement is* Δh_2 *– the maximum deviation from the threshold within the time interval. Figure 4.9(b) plots the signal of* (Noise − 50) *under option (ii) where the robustness value of violating the requirement is* Δh_3*. Option (i) has a better performance than option (ii) in terms of the robustness value of violating the requirement, because* $\Delta h_2 = 50$ *and* $\Delta h_3 = 150$*. However, a decision maker may actually find option (ii) a better resolution, because the noise level only exceeds the threshold 50 db for a shorter period of time (2 min instead of 12 min). Thus, smaller value of robustness violation degree sometimes does not imply a better resolution.*

To address this limitation, we present two new metrics for measuring the degree of violating smart city requirements specified in STL: (1) the percentage of time when a requirement is violated, and (2) the integral of signal deviations. To start with, we define Equations 4.6 and 4.7 to calculate the positive part (denoted by $\theta^+(x)$) and the negative part (denoted by $\theta^-(x)$) of a function $f(x)$, respectively.

$$\theta^+(x) = \max(f(x), 0) = \begin{cases} f(x) & \text{if } f(x) > 0 \\ 0 & \text{otherwise.} \end{cases} \tag{4.6}$$

$$\theta^-(x) = \min(f(x), 0) = \begin{cases} f(x) & \text{if } f(x) < 0 \\ 0 & \text{otherwise.} \end{cases} \tag{4.7}$$

We compute the percentage of time when a given signal x violating $\square_{(a,b)}$ $(x < \lambda)$ as follows:

$$\eta = \frac{1}{b-a} \int_{\tau+a}^{\tau+b} \text{sgn}(|\theta^-(x(t) - \lambda)|)dt. \tag{4.8}$$

We use Equation 4.9 to compute the integral of signal deviations accumulated in a period when the requirement $\square_{(a,b)}$ $(x < \lambda)$ is violated.

$$\gamma = \int_{\tau+a}^{\tau+b} (|\theta^-(x(t) - \lambda)|)dt. \tag{4.9}$$

Example 4.4. *Figure 4.9(c) and Figure 4.9(d) shows the percentage of time when requirement* $\square_{(0,25)}$ (Noise < 50) *is violated under option (i) and (ii), respectively. Option (ii) has a better*

performance with this metric of requirement violation degree, because $\frac{(t_1+t_2)}{25} < \frac{t_3}{25}$.

Figure 4.9(e) and Figure 4.9(f) shows the integral of noise level deviations when requirement $\square_{(0,25)}\ (\mathsf{Noise} < 50)$ *is violated under option (i) and (ii), respectively. The integral value for option (i) is the sum of areas* $S_1 + S_2$*, while the integral value for option (ii) is* S_3*. It turns out that* $S_1 + S_2 = S_3 = 26$*. Thus, option (i) and option (ii) have the same performance with this metric of requirement violation degree.*

These three metrics are useful to evaluate different smart city requirements. Robustness value is more suitable for requirements with hard constraints, such as the emergency vehicle waiting time and the number of accidents. Most city requirements are soft constraints, which do not require the signal strictly within a threshold bound. In such cases, the percentage of violation time is an important measurement, especially for environmental signal like the pollution level. The integral metric combines robustness and violation time. Therefore, it helps to compare the overall performance between different signals. For example, calculating the integral of violating requirements in the transportation domain (e.g., the waiting number and waiting time of vehicles) helps to reveal the congestion degree.

4.4 Evaluation

CityGurad and CityResolver are evaluated using a smart city simulator, which is extended from SUMO, a transportation simulator. The evaluation uses a real map of one-half of Manhattan, New York City. The transportation state is generated from the Traffic volume counts of New York city data [100]. This data set contains the traffic volumes from 160 streets in Manhattan during 2013-2014. Analyzing the data set, the average traffic volume on all streets is 105,397 vehicles and 658 vehicles per street per hour, making the in-coming vehicle rate at 5.5 second per vehicle.

We implement a smart city simulator using the data set and the Simulation of Urban MObility (SUMO) [99], a transportation simulator which allows modelling of traffic systems including road vehicles, public transport and pedestrians. We build ten smart services (see Table 4.6) from the domains of transportation, emergency, and environment running over 140 locations of lower Manhattan, New York, as shown in Figure 4.2. Due to the limitations of SUMO, only services from the above domains are implemented. However, these are examples of the most common smart services running in real cities and are both representative and important. We verify resolution options with the same city

Table 4.1: Services running in Simulated Manhattan

No.	Service	Domain	Description
S1	Congestion Service	Transportation	Its purpose is to minimize traffic congestion. When the waiting number of vehicles on the lane exceeds 50% of its total tolerance, it will adjust the traffic signal to release congestion.
S2	Pedestrian Service	Transportation	Its purpose is to minimize waiting time of pedestrians. When more than 2 pedestrians press crossing button, it will shorten their waiting time by adjusting traffic signals.
S3	Vehicle Navigator	Transportation	Its purpose is to release vehicles from traffic congestion. When there is a traffic congestion or closed lane causing a vehicle waiting for a long time, it will call the re-route function to choose the next shortest path to its destination.
S4	Air Pollution Control	Environment	Its purpose is to control air quality level with emphasis on the CO and HC gas released by vehicles on streets. It will limit the number and speed of vehicles when air pollution level is high by adjusting traffic signal and sending speed request to vehicles directly.
S5	PM2.5/PM10 Control	Environment	Similar to S4 with emphasis on the PM2.5 and PM10 in the air.
S6	Waste Management	Environment	Its purpose is to manage waste in cities by sending out waste pickup vehicles regularly.
S7	Noise Control	Environment	Its purpose is to control noise pollution causing by traffic. When noise level exceeds its threshold, it will control the number of vehicles going through related streets and redirect vehicles on the streets by adjusting traffic signals.
S8	Event Service	Environment	Its purpose is to ensure smooth operation of a city event by blocking the lanes nearby the event.
S9	Accident Service	Emergency	Its purpose is to take the first action to block the adjacent areas of a traffic accident.
S10	Emergency Service	Emergency	Its purpose is to minimize the waiting time of emergency vehicles. When there is an emergency vehicle waiting in the lane, it will adjust the traffic signal to let it go through immediately.

safety and performance requirements in detection component, which are listed in Table 4.2. Different types of requirements are suitable to different space ranges and distributed over even more than 140 locations in the map. We use CityGuard to detect conflicts and CityResolver to provide resolutions for those conflicts. To generate resolution options, CityResolver uses the Gurobi optimization tool [101] to solve integer linear programs.

The experiments are evaluated on a server machine with 16 core CPUs; each core is 3.1GHz. The operating system is Ubuntu 14.04.5.

4.4.1 Initialization and Metrics of Simulation

In order to simulate the performance of CityGuard in a real city environment, to start with, real city data from Manhattan is analyzed. From Traffic volume counts of New York city data collected by DOT for the New York Metropolitan Transportation Council [100], the traffic volumes from 160 streets in Manhattan during 2013-2014 are obtained. It is calculated from the data set that the average traffic volume for all streets is 105,397 vehicles and 658 vehicles per street per hour, making the in-coming vehicle rate as 5.5 seconds per vehicle.

In order to generate a scenario closest to the real traffic pattern in Manhattan, three steps are performed to configure the simulation. First, we selected the average traffic volume data of Manhattan

Table 4.2: List of Requirements (The check mark indicates the suitable space range of the requirement in practice.)

Requirement	Intersection	Street	Block
R1: The noise level in the school area should always be less than 50db.	✓		✓
R2: The CO emission in an intersection should always be less than 40mg.	✓		✓
R3: The waiting time of the traffic in an intersection should not be greater than certain threshold λ.	✓	✓	
R4: An emergency vehicle should wait in an intersection for more than 10 seconds.	✓		
R5: No vehicle collision should occur.	✓	✓	
R6: The number of vehicles in a street should never exceed its maximum vehicle capacity.		✓	
R7: The traffic yield number in a street should not increase by certain threshold λ.		✓	
R8: The number of pedestrians waiting in an intersection should not be greater than certain threshold λ.	✓		
R9: Emergency vehicles should not be directed to a blocked lane or area.		✓	
R10: The noise level in a street should always be less than 70 dB.		✓	✓
R11: The hydrocarbons (HC) emission in a lane should always be no more than 1 mg.		✓	✓
R12: The particulate matter (PMx) emission in a lane should always be no more than 0.2 mg.		✓	✓

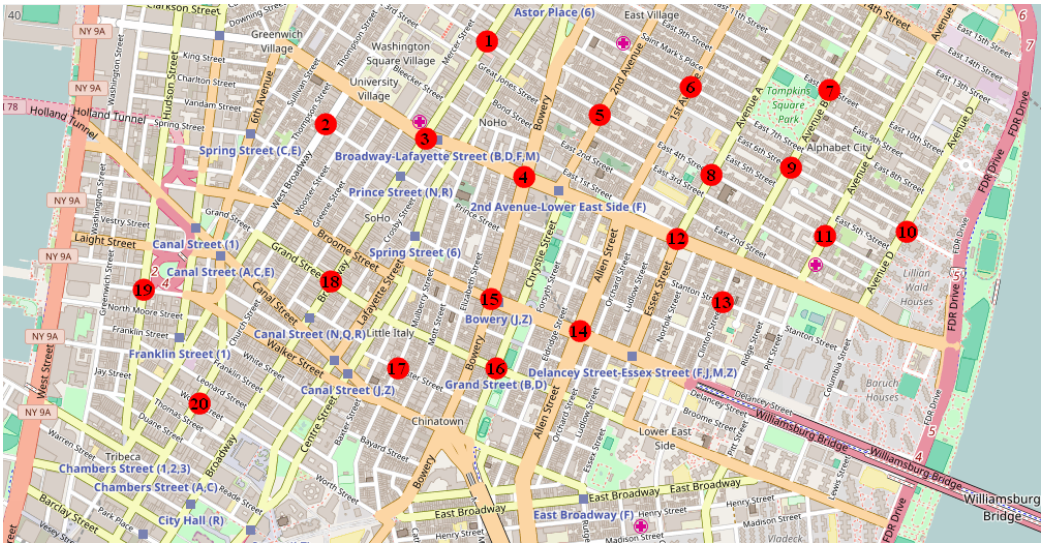


Figure 4.10: Simulated Manhattan with 10 services running at 20 representative different locations (denoted as red points).

from 8:00 am to 2:00 am to generate the traffic data stream in the simulation. Second, the traffic streams for main streets of New York city, such as the Bowery, Allen Street, and Broadway, are set based on their own average traffic volume per street. Finally, stream data for other streets follow the average volume for the entire Manhattan area, i.e. the in-coming rate of 5.5 seconds/vehicle is used. In this way, the lower half of Manhattan including 102 streets and 454 traffic lights are used as the **platform** for all simulations in the evaluation.

Furthermore, important safety and performance **metrics** are chosen for evaluation, as shown in

Table 4.3: Metrics for Evaluation of City Performance: S=Safety and P=Performance Metrics

Name	Description
Jam (P)	Number of cases when a vehicle can not continue because there was no space on the next lane
Yield (P)	Number of cases when a vehicle is unable to cross an intersection where it did not have priority
Collision (S)	Number of cases when a vehicle violated its minimal distance requirement in relation to its leader vehicle
Wrong Lane(S)	Number of cases when a vehicle was unable to move because it could not continue its route on the current lane and was unable to change to the correct lane
Mean Speed (P)	The mean speed of the vehicles on the specific lane (km/h)
Waiting Number (P)	The number of vehicles waiting on the lane, a speed of less than 0.1 m/s is considered a wait.
Waiting Time (P)	The time that a vehicle waits on the lane
Noise (S)	The noise emitted by the vehicles on the specific lane (dB)
CO (S)	The complete amount of CO emitted by the vehicles on this lane during the actual simulation step (mg)
HC (S)	The complete amount of HC emitted by the vehicles on this lane during the actual simulation step (mg)
PM_x (S)	The complete amount of PM_x emitted by the vehicles on this lane during the actual simulation step (mg)

Table 4.3. The first four metrics are obtained from internal models of SUMO, indicating the transportation safety and performance. For instance, the possibility of a collision happening increases when the density of traffic increases and the distance between vehicles shrinks. Meanwhile, these metrics when applied to emergency vehicles also indicate the performance of the emergency domain. Moreover, mean speed, waiting number and waiting time per lane are measured for transportation performance. Noise, CO, HC and PM_x are measured for environmental performance and are considered as safety metrics.

For the evaluation, the requirements shown in Figure 4.4 are assumed to be specified for the smart city.

To better understand the complexity and scope of conflicts, the pre-processing component is first evaluated in isolation. This demonstrates the spatial and temporal effects of individual service. Next, the overall CityGuard is evaluated.

4.4.2 Pre-processing

In the Pre-processing component, single actions are intercepted and their primary and secondary effects on the environment are simulated to see if there is any violation of safety or performance requirements.

Spatial and temporal ranges of action effects are tested by CityGuard SUMO on 10 services in 20 locations. At first, the city is simulated without services and the metrics for all the streets near the services are recorded as the baseline. Each service is simulated individually in different locations,

and the same states are recorded and compared with the baseline. If the variance is above 5%, it is viewed as an effect on the streets from this action. In this way, the number of blocks away from the service block that are affected by the action is identified for each service in all locations. Similarly, how long the effects last on the environment are also monitored.

The results are shown in Table 4.5, the first column are the metrics, the second column lists services running with and without CityGuard, the third column is when there are no services at all, and S1 to S10 are the data from just one service running. Following insights are obtained from the results that are presented Table 4.5.

When one service improves one aspect of city, its secondary effects may have a negative influence on other metrics. If not controlled, this influence may exceed the safety and performance requirements.

When there is no CityGuard, actions from 5 services cause collisions, which affect city safety significantly and even create more serious secondary effects. These collisions are prevented by CityGuard. Meanwhile, the number of jam and yield violations also exceeds the threshold of safety requirements, which are highlighted in Table 4.5. However, with control of CityGuard, jam and yield from all actions are controlled under the safety requirement.

Another key metric for safety performance, waiting time of emergency vehicles (Row 8), is increased significantly by 8 services , exceeding the threshold of safety requirements. With CityGuard it is controlled to under 10s. For example, waiting time of emergency vehicle is reduced from 19.3s to 9.3s by CityGuard, improving the performance by 101%.

It is important to notice, that in these 8 cases, comparing the performance of other metrics when CityGuard runs with no services in the city, it improves the city's performance, i.e., services' functions are not affected by CityGuard. For example, air pollution service (S4) increases the waiting time of emergency vehicles to 15s without CityGuard and to 10s with CityGuard, and where, CO are 7.8mg and 7.6mg, respectively. Comparing with 11.74mg of CO release without any service, S4 with CityGuard improves the air quality. As a result, with CityGuard, air quality is improved without affecting emergency vehicles. In some cases, service performance on one or two metrics is compromised with CityGuard because some of its unsafe actions are rejected. However, CityGuard's role here is to obtain a safe environment while helping services improve the city. For example, in column S1, waiting time for normal vehicles is 98.5 seconds without CityGuard, while 100.2 with CityGuard. Though this

Table 4.4: City Safety Requirements

Transportation:

- Actions should not cause collisions of vehicles.
- Vehicles should not be directed to travel in the wrong direction or to blocked roads.
- Traffic signal lights should follow safety logic.
- Vehicles from orthogonal directions should not cross an intersection simultaneously.
- Actions should not increase traffic congestion by more than 10% .
- Actions should not increase Yield by more than 15% .
- Actions should not increase waiting time of emergency vehicles by more than 10%
- The number of waiting vehicles in a lane should be less than the maximum vehicle capacity of the lane.

Emergency:

- Emergency vehicles should not wait for more than 10s at a intersection.
- Emergency vehicles should not be directed to a blocked lane or area.

Environment:

- Action should not create more than 50 dB noise per lane.
- Action should not emit more than 50 mg CO per lane.
- Action should not emit more than 1 mg HC per lane.
- Action should not emit more than 0.2 mg PMx per lane.

performance is compromised by 2 seconds, it still improve the transportation performance comparing with the one without service (121.82 seconds). Most importantly, the waiting time for emergency vehicles decreases from 11.5 to 10 comparing the results without and with CityGuard, consequently S1 is controlled to work under safety requirements.

If a service does not violate requirements at all, it will not be affected by CityGuard, such as S3 and S6.

Some additional observations from these simulations are:

- The ranges of spatial and temporal effects vary by functions of services and locations. For example, Event Service (S8) usually has a longer effect time than the Congestion Service (S1). Effect range at Location 15 is always larger than that at Location 20.

- The effects are more significant on the streets of the intersection where services run, rather than the neighboring streets.

- The Accident Service (S9) has the largest average spatial effect range of up to 5.6 blocks while the Noise Service (S7) has the lowest one of 0.8 blocks. The average effect range for all 10 services is 2.3 blocks.

Table 4.5: Effects on City with single services running

Metrics		No S	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10
Jam	No CG	102	100	**422**	60	102	68	102	84	**402**	**222**	**290**
	City-Guard	-	100	**110**	60	102	68	102	84	**86**	**120**	**120**
Yield	No CG	192	230	**400**	178	226	190	194	210	**622**	**530**	**598**
	City-Guard	-	218	**212**	178	200	190	194	198	**202**	**196**	**218**
Collision	No CG	0	0	**4**	0	**2**	0	0	0	**4**	**4**	**6**
	City-Guard	-	0	**0**	0	**0**	0	0	0	**0**	**0**	**0**
Wrong Lane	No CG	22	22	40	6	14	12	22	20	**82**	42	**46**
	City-Guard	-	20	20	6	14	12	22	20	**20**	20	**20**
Mean Speed	No CG	11.28	12.36	10.29	13.64	9.8	10.1	11.28	12.21	8.98	9.14	9.11
	City-Guard	-	13.22	11.1	13.64	10.6	11	11.28	13.5	10.89	10.7	8.99
Waiting Number	No CG	1.66	0.71	**5**	0.72	2.1	2.21	1.67	1.7	**4.71**	**3.98**	1.89
	City-Guard	-	0.73	**1.7**	0.72	1.8	1.91	1.67	1.65	**1.69**	**1.8**	1.86
Waiting Time	No CG	121.82	98.60	**311.7**	101.5	**150.1**	**149.4**	121.9	136.1	141.3	157.6	181.4
	City-Guard	-	100.2	**130.97**	101.5	**129.41**	**134.87**	121.9	123.1	131.8	137.2	134.7
E Waiting Time	No CG	9.5	**11.50**	**13.3**	**10.93**	**15**	**13.5**	9.5	**15.3**	**17.1**	**19.3**	3.2
	City-Guard	-	**10**	**9.4**	**10**	**10**	**9.6**	9.5	**10**	**9.8**	**9.6**	3.41
Noise	No CG	33.34	34.98	40.12	32.1	31.5	32.5	33.34	17.6	37.6	39.6	39.4
	City-Guard	-	32.14	35.6	32.1	30.93	31.1	33.34	18.7	35.6	35.1	39.6
CO	No CG	11.74	10.85	**16.9**	10.65	7.8	7.41	11.74	11.61	13.4	14.11	13.8
	City-Guard	-	10.75	**12.4**	10.65	7.6	7.31	11.74	10.71	12.31	13.13	12.3
HC	No CG	0.49	0.46	0.71	0.48	0.23	0.29	0.49	0.51	**0.69**	**0.91**	0.65
	City-Guard	-	0.46	0.53	0.48	0.22	0.27	0.49	0.46	**0.64**	**0.69**	0.56
PMx	No CG	0.08	0.07	0.12	0.69	0.05	0.03	0.08	0.08	0.11	**0.17**	0.11
	City-Guard	-	0.07	0.09	0.69	0.05	0.03	0.08	0.08	0.1	**0.13**	0.09

- Temporal effects of different services vary significantly not only by service type and locations, but also by the specific contexts of the events. For example, the effect range of services is larger in area and longer in time when the traffic density is heavier.

4.4.3 Environmental Conflict Detection

In the environmental conflict component, integrated effects of concurrent actions are tested in simulated Manhattan. The relationship between environmental conflicts detected and the number of services running in the city is analyzed. N services are randomly chosen for 50 times and average performance is recorded. Performance of representative metrics for safety and performance, i.e. number of collisions, number of jams, CO air pollution, and traffic waiting time per lane are shown in Figure 4.12, leading to the following observations.

- Generally, with the growing number of services running in the city, effects on the environment become worse if there is no safety protection mechanism.

(1)Without CityGuard (2)With CityGuard

Figure 4.11: Performances of traffic at the intersection between Bowery and Kenmare with and without CityGuard. Yellow and red objects denote regular and emergency vehicles. With CityGuard, there are higher traffic flow, less congestion, and emergency vehicles are prioritized for faster travel time.

- The number of collisions increase significantly when more than 7 services are running together, which seriously violates city safety. However, there is no collision with CityGuard.
- Similar to the number of the collisions, the number of Jams is also affected by the number of services running, which can be controlled under a performance threshold with CityGuard running.
- Although waiting time is not affected as drastically as other metrics, it is seen from Figure 4.12 (4) that CityGuard can help the city to obtain a relatively stable performance with increasing number of services. When the number of services is 10, the waiting time is improved by 35%.
- CityGuard improves Air quality (as measured by emission of CO) over services without CityGuard by 51.3% when 5 services are running, and by 73.7% when 10 services are running.

The total number of conflicts detected are summarized by 20 locations of services in Figure 4.13. In all locations, environmental conflicts have the highest percentage, which usually is at most twice the percentage of device conflicts and as much as 4 times as single unsafe actions. Moreover, the number of conflicts vary from location to location. The busier and larger an intersection is, the more conflicts occur there.

4.4.4 Option Generator

The total number of resolution options increases exponentially with the number of actions. That is, given n actions, there are 2^n possible choices of resolutions. CityResolver builds an ILP-based

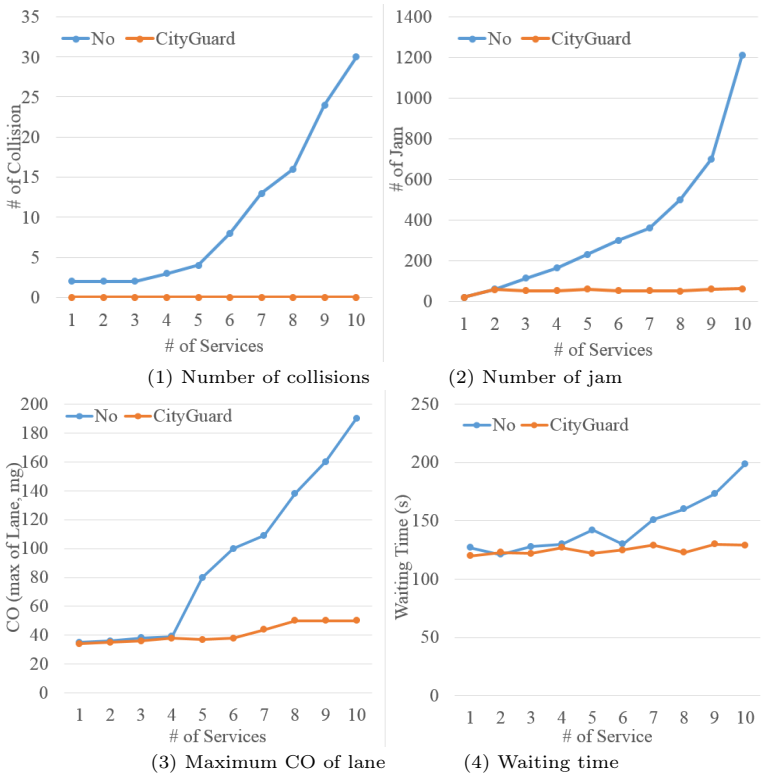


Figure 4.12: Comparison of Service Effects on Environment with and without CityGuard

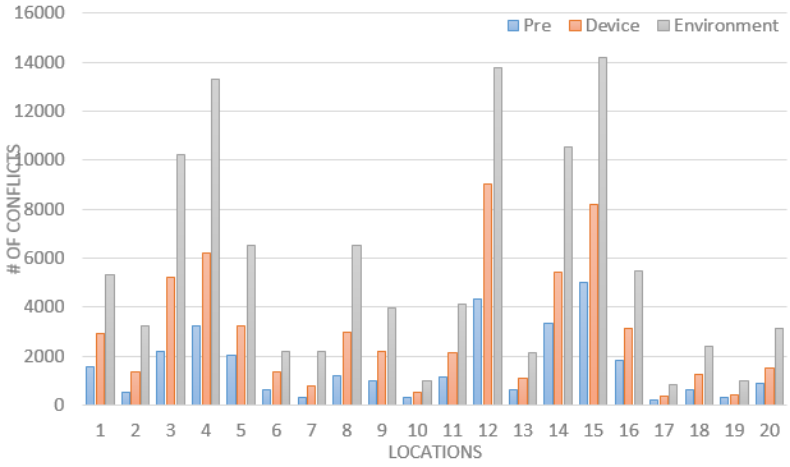
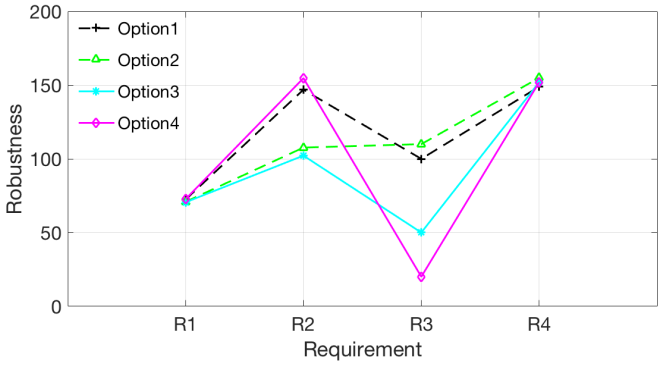
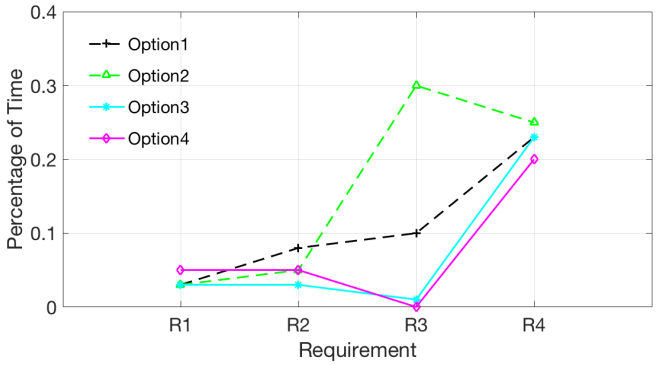


Figure 4.13: Number of conflicts detected from each location

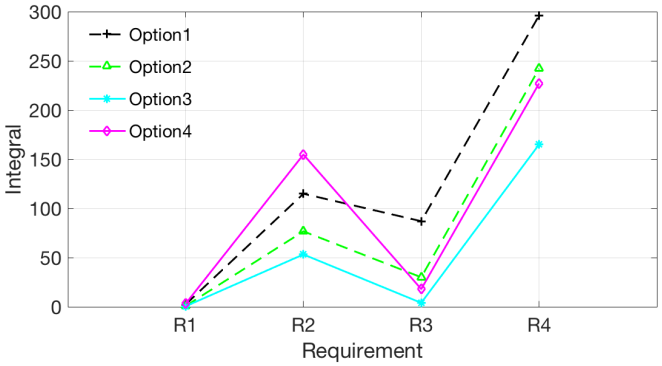
option generating model, which is able to identify a set of optimal options regarding the sum of action requests multiplied by their weights. We tested the option generator for over 500 conflicts and where each conflict has up to 100 actions. The computation time of generating resolution options for each conflict is within 1 second using CityResolver.



(a) Metric: Robustness violation value



(b) Metric: Percentage of violating time



(c) Metric: Integral of signal deviations

Figure 4.14: Trade-offs of four options based on their performance on requirements R1-R4

4.4.5 Computing Violation Degree

CityResolver provides the three metrics for calculating the violation degree. For example, in this study we measure the violation degree on 4 requirements (R1 - R4) using the three metrics. The traces of each option are generated from the city simulator. When predicting the future city states, we incorporate the known disturbance factors. For example, one of our results shown in Figure 4.14

considers that rush hour is coming, which will cause heavy traffic. Therefore, in our simulation, we increase the traffic volume at that future time to obtain a more accurate prediction. We observe the following based on Figure 4.14.

- There are trade-offs between different options using all three metrics. For example, in (a), Option 4 has the best performance on R3 (i.e., the congestion requirement), but has the worst performance on R2 (i.e., the noise requirement). The other three options, though not best on all requirements, have a more even distribution.

- Different metrics lead to different results: Option 4 is the best in (a) and (b) regarding requirement R3, but is the second best in (c).

- For the robustness metric (a) and integral metric (c), it is difficult to compare the performance of the same option on different requirements, because measurement scales are not necessarily the same. However, it is uniform using the percentage of time (b), where all violation degrees are on scale of 0 to 1.

- All three metrics can help a decision maker to compare the performance of different options on the same requirement. They may choose the metric based on the requirement type and the context.

- It is helpful to compare more than one metric, because they represent different properties of the requirement. For example, Option 4 has a worst performance on R2 measured by the robustness and integral metrics, but does not have significant violation on the percentage of time metric. The results also indicate that Option 4 exceeds the noise level threshold for only a few times within this period, but each time the difference is large.

4.4.6 Overall Performance

We evaluate the city performance with CityResolver across domains of transportation, environment and emergency. Specifically, the performance evaluation uses metrics including the number of violated requirements per conflict, emission of CO, noise, the waiting time for the emergency vehicle, the waiting number of vehicles, and the waiting time for pedestrians. The number of violated requirements indicates that when detecting one conflict or verifying a solution, how many requirements are violated by this conflict or option. The remaining metrics are calculated by the average value at one intersection.

Table 4.6: List of Services for Overall Evaluation

Service	Description
S1: Traffic Service	It controls traffic signals in street intersections to relieve congestion and optimize or improve traffic performance.
S2: Emergency Service	It requests green traffic signals in order to transport patients in critical conditions to hospitals as soon as possible.
S3: Accident Service	It blocks a street where some accident occurs and alert nearby vehicles to detour.
S4: Infrastructure Service	It schedules infrastructure check-up and repair appointments.
S5: Pedestrian Service	It shortens the pedestrians' waiting time by adjusting traffic signals when pedestrians wait in the intersection.
S6: Air Pollution Control	It adjusts the traffic by adjusting traffic signal and sending speed request to vehicles when CO emission is high.
S7: PM2.5/ PM10 Control	It adjusts the traffic when PM2.5/ PM10 emission is high by adjusting traffic signal and sending speed request to vehicles directly.
S8: Parking Service	It directs the driver to the nearest parking lot.
S9: Noise Control	When noise level exceeds its threshold, it controls the number of vehicles going through related streets and redirect vehicles on the streets by adjusting traffic signals.
S10: Event Service	It ensures operation of a city event by blocking the lanes nearby the event.

Table 4.7: Comparison on the city performance with CityGuard and with CityResolver

Case	System	Number of Violated Requirements	CO (mg)	Noise (db)	Emergency Waiting Time (s)	Vehicle Waiting Number	Pedestrian Waiting Time (s)
Case 1: [S1, S2]	None	20	53.12	67.73	21.00	23	65.00
	CityGuard+Pri	7	67.90	72.42	**6.70**	32	60.00
	CityResolver	**0**	**39.70**	**46.54**	9.80	**22**	**35.90**
Case 2: [S1, S2, S3]	None	28	55.20	49.00	31.20	19	83.00
	CityGuard+Pri	18	54.20	62.00	11.00	21	88.00
	CityResolver	**2**	**44.30**	**48.90**	**8.70**	**15**	**63.50**
Case 3: [S1, S8, S10]	None	16	53.91	67.00	9.20	23	53.20
	CityGuard+Pri	**0**	49.20	48.70	8.80	24	50.10
	CityResolver	**0**	**30.80**	**40.80**	**7.80**	**18**	**49.20**
Case 4: [S6, S7, S9, S10]	None	15	48.30	59.00	14.50	42	79.20
	CityGuard+Pri	8	**38.60**	62.10	13.80	34	76.30
	CityResolver	**1**	39.50	**56.30**	**8.30**	**29**	**65.20**
Case 5: [S2, S3, S4, S5, S9]	None	39	87.30	45.00	7.40	39	68.30
	CityGuard+Pri	32	90.30	46.00	7.80	39	65.70
	CityResolver	**4**	**62.70**	**43.00**	**6.70**	**28**	**59.60**

For example, if the effective range of one conflict is 3 blocks (i.e. involving 9 intersections), then the value is the average value of these 9 intersections.

We compare the result with two baselines: (1) a smart city without any controller for conflict resolution, and (2) CityGuard. To the best of our knowledge, there is no other existing solution for conflict resolution in smart cities. CityGuard focuses on the conflict detection and implemented a simple priority-based conflict resolver, i.e. when there is a potential conflict between different action requests, it only accepts the action with the highest priority. Also, it does not verify the performance of selected actions. The experiments are conducted on the lower Manhattan with 10 smart services running over 140 locations. The results from 5 cases are shown in Table 4.7, from which we have the

following observations.

Case 1: Conflict happens between the S1 smart traffic service and the S2 smart emergency service. Without resolution, it violates 20 city requirements. With CityGuard, it detects the conflict and accepts the actions of the emergency service, which reduce the number of violated requirements and improve city performance regarding the emergency vehicle waiting time. However, it harms the performance on CO and noise with a bias in favor of the emergency. Instead, CityResolver finds an optimal solution, which reduces the number of violated requirements to 0. Compared with CityGuard, CityResolver also improves other metrics, for example, it reduces the CO emission by 41.5%.

Case 2: Another conflict happens among S1 (traffic service), S2 (emergency service) and S3 (accident service). Similar to Case 1, both CityGuard and CityResolver propose a resolution and improve the city performance. Two things to be noted, 1) CityResolver has a better overall performance than CityGuard, 2) CityResolver also does not find a resolution satisfying all requirements, but it reduces the violation number by 13 times and 8 times compared to the none-control system and CityGuard, respectively.

Case 3: It demonstrates a conflict among traffic, parking and event services, where both CityGuard and CityResolver find a resolution that satisfies all requirements. However, CityResolver still has better performance, for example, it beats CityGuard by 37.4% on CO and 25% on the vehicle waiting number. The reason is that CityResolver accepts more actions if possible, which as a result, benefits the city.

Case 4 and *Case 5* evaluate cases under a larger number of services and actions. CityResolver finds potential options and verifies them within a reasonable time. In particular, it maintains a small number of violated requirements comparing to baselines. By showing trade-offs between the requirements that have to be violated, CityResolver gives the city manager a chance to select a better choice based on the context. Meanwhile, though not largely, CityResolver improves performance regarding each domain by 6.5% to 30.6% comparing to CityGuard.

In summary, CityResolver reduces the number of violations significantly and improves the overall city performance without sacrificing the performance on other metrics. However, to be noted, CityResolver does not always have the best performance over CityGuard. For example, in Case 1 and Case 4, its performance on the emergency vehicle waiting time and CO emission are worse than CityGuard. The reason is that the emergency service and air quality service have a higher priority and so do

their actions with CityGuard resolution. With this bias, it does not have good performances on other metrics. On the contrary, CityResolver values high-weight actions as well as optimizes other metrics.

4.5 Discussion

Safety-aware conflict detection and resolution are critical and complicated issues in smart cities, which is difficult to be solved at once. CityGuard is a watchdog solution to dynamically improve the safety of smart cities by focusing on actions that impact the environment and act across services. It does not and we argue fundamentally, cannot apriori guarantee complete safety due to the dynamics and uncertainty in the real world. It is a general architecture which can be integrated into any Smart City IoT platform. Being the first work of its kind, CityGuard has limited scope in the following issues.

Safety Requirements: In this dissertation, it is assumed that actions and requirements that are integrated with CityGuard have a uniform format. For example, it is assumed that actions provide necessary information, i.e., device, act, duration, and pre-condition(s). The mapping of actions to the safety and performance requirements were performed manually, because the existing safety rules of cities are defined in English by different government departments. There are no tools to translate them into code automatically. However, this manual mapping process doesn't affect the generalizability of CityGuard. Also, if new requirements arise, the module called *City Safety & Performance Requirement* can update the requirements maintained in CityGuard and detect conflicts using the new requirements (See Figure 4.4). In the future, we will explore ways to specify requirements and algorithmically map actions to requirements. In this case, semantic token extraction techniques for textual interventions [15] can be useful.

Conflict Detection and Safety: Simulating the primary and secondary effects of an action, CityGuard checks the effects with the city-wide safety requirements to check if there is a conflict among actions. Many such conflicts are found and resolved. However, some smart services have their own internal safety requirements not directly visible to CityGuard. CityGuard cannot always detect the safety violation of internal safety requirements of a service unless they impact the environment in a negative manner as specified in the city-wide requirements. Therefore, although currently CityGuard's goal is to significantly improve the safety of a city, safety violations can still occur. Ideally, in the

future, as services are added to a city, all the internal safety requirements should be exported and compared to the city wide requirements. Any conflicts here should also be resolved.

Conflict Resolution: Currently, CityGuard focuses on the methods for conflict detection rather than providing a comprehensive solution for conflict resolution. Only predefined priority based resolvers are used to resolve conflicts once they are detected. However, in real scenarios, conflict resolutions can be much more challenging and might require considering several contextual factors. The list of such factors includes, but is not limited to, the importance of the action (i.e., that causes conflict) and its service, the effects on the environments and human beings, the cost of rejection, the optimal combinations of conflicting actions, dynamically changing objectives, etc. Thus, a comprehensive solution of conflict resolution demands further research.

Simulation: In the current implementation of CityGuard, the selection of applications and the accuracy of conflict detection rely on the accuracy of the SUMO simulation and its embedded models. This is reasonable as (i) SUMO has been widely used to provide simulations on city mobility with sophisticated physical models developed in research and used in practice for over 15 years; (ii) SUMO is only an example tool for CityGuard, which can be updated, combined, or replaced with more sophisticated models/simulators once they are available; (iii) The evaluations in this paper are limited yet realistic to illustrate how we detect conflicts and improve a city's safety and performance. The approach for conflict detection in CityGuard is sophisticated enough to be applied in other domains once accurate models of those domains are available.

Human Factors: When CityGuard simulates the potential primary and secondary effects of actions during its assessment phase, it assumes that the people (i.e., citizens) and the devices comply with the recommended actions. Decisions to accept or reject actions are based on these assumptions. However, the city state evolves based on what actually happens in the city, e.g., a person may not follow the advice or an unexpected / unprecedented event may occur. CityGuard includes a feedback loop to monitor and react to this situation.

Privacy and Security: Privacy and security are of great significance to a smart city. As they are out of the scope of this paper, a brief discussion below highlights the key issues.

There are potentially many different privacy policies in the smart services and for the city as a whole. Policies themselves might conflict with each other or be violated due to actions from services. When actions are taken by services, CityGuard can be extended to match those potential actions with city

wide privacy policies and, therefore, detect potential policy conflicts, dynamically. In other words, the core concepts found in CityGuard can be applied to privacy. If services export their privacy policies then conceptually a general policy conflict detection between the city and a service can be detected at installation time.

There are two main levels of security issues in the context of this work, which are the security of smart services and the security of CityGuard itself. The security of smart services should be maintained by themselves. However, if a service has been attacked and it is taking erroneous actions, CityGuard helps in avoiding some safety violations because those erroneous actions might conflict with the safety requirements known to CityGuard and therefore be blocked. In addition, there are ways that CityGuard itself can be attacked, such as approving unsafe actions, blocking actions from other services, setting higher priority for the service, etc. Therefore, security mechanisms must be included in CityGuard prior to real deployment.

4.6 Summary

CityGuard, a safety-aware novel watchdog architecture for detecting and resolving conflicts in a smart city is developed and evaluated. CityGuard is safety-aware as it focuses on maintaining safety requirements by detecting and resolving conflicts before they occur. In doing so, CityGuard also considers the context of potential conflicts and resolves conflicts to maximize performance metrics. CityGuard is able to simulate the primary and the secondary effects of the actions performed by services, detect and resolve conflicts among those actions, and thus improve safety. Evaluations are performed using a simulation of part of New York City with 10 smart services located in 20 places. Using 6 safety metrics and 5 performance metrics, the evaluation shows that CityGuard is able to minimize safety violations, and improve overall city performance when compared to baseline methods.

In this chapter, we build CityResolver – a decision support system for conflict resolution in smart cities. Using an Integer Linear Programming based method, CityResolver first generates a small set of potential resolution options considering the dependency of actions and adaptive policies of their services. It verifies the set of options with an STL based approach and calculates the effects of options on different requirements taking the disturbance factors into account. Then it shows the trade-offs between resolution options in a dashboard supporting decision makers to choose the best resolution. The evaluation results show that, CityResolver is able to reduce the number of violation

requirements significantly comparing to the smart city without a controller and with a priority based resolver. For example, CityResolver reduces the number from 39 down to 4, while CityGuard reduces the violation number from 39 to 32. Moreover, CityResolver improves city performance comparing to the baselines. For example, in our experiments, it beats CityGuard on the CO emission up to 37.4% and the congestion by 25%.

Chapter 5

Logic-Enhanced Learning

Deep Neural Networks (DNNs), especially Recurrent Neural Networks (RNNs) have great achievements for sequential prediction tasks and are broadly applied to support the decision making of Cyber-Physical Systems (CPSs) [102, 103, 104]. Usually, in CPSs, RNNs are applied to predict the changing of system states or their environment. Systems take actions based on the prediction to guarantee the safety and performance of the system. For example, the power plant predicts the usages of energy in the next few days and decides how much energy to generate. An event service predicts the population and traffic for a big concert and allocates police and security resources. Training RNNs for complex CPSs (i.e., creating a prediction model) such as for Smart Cities is difficult [105]. The models are not always robust, often subject to anomalies, and subject to erroneous predictions, especially when the predictions are projected into the future (errors grow over time).

On the other hand, the target sequence often follows specific model properties or patterns, which should also be followed by the predicted sequence. For example, power plants have maximum and minimum limits of energy that can be generated per day, and the changing of air quality is relevant to the changing of traffic volume in the past hour. However, RNNs have no way to guarantee that their estimated distributions satisfy these model properties, especially for the properties with multiple variables and temporal features. Failure to follow these model properties can result in inaccurate and even meaningless results, e.g., predicted traffic volume exceeding the road capacity and inaccurate population estimation due to ignorance of big events happening a few hours ago.

It is very challenging to enforce multivariate RNNs to follow temporal model properties in a sequence

prediction task. The optimization mechanism of RNNs (i.e., back propagate the loss between estimated value and target value individually in a predicted sequence at each time unit without comparing the temporal correlation of the two sequences, thus lack an integrated view about the sequential predictions) causes the challenge of the networks to follow the temporal model properties. In addition, unlike classification problems, it is more difficult to find an alternative approximate sequence that satisfies the property for knowledge distillation.

Contributions: In this chapter, we create a new temporal logic-based learning framework, called STLnet, to guide the RNN learning process with auxiliary knowledge of model properties and to produce a more robust model that can then be used for improved future predictions. Unlike existing approaches, STLnet enforces the predicted multivariate sequence to follow its model properties by treating the sequence (i.e. a trace) as a whole. We first identify six key types of model properties and formalize them using Signal Temporal Logic (STL) [106]. Following the idea of knowledge distillation [32], the STLnet framework is built with a teacher network and a student network. In the teacher network, we create a STL trace generator to generate a trace that is closest to the trace predicted by the student network and satisfies the model properties simultaneously. We also create algorithms to efficiently generate satisfaction traces tailored to deep learning processes. We evaluate the performance of STLnet by applying it to an LSTM [107] network and a transformer network [108] for multivariate sequential prediction. The experimental results show that STLnet significantly increases the satisfaction of different types model properties (by about 4 times) and further improves the prediction accuracy (by about 18.5%).

To the best of our knowledge, our framework is the first work that integrates signal temporal logic with deep neural networks for sequential prediction. The strength of temporal logic offers a much stronger power to control a sequential system and a more flexible way to specify various types of properties. Different from previous literature, our method also creates a practical way to ensure the satisfaction of the logic rules. STLnet can be applied to general deep models to perform multivariate time series prediction and can be trained in an end-to-end manner. STLnet increases the robustness of the deep learning models.

5.1 Model Property Formalization using Signal Temporal Logic

In this chapter, we refer to model properties as the inherent properties, rules, or patterns followed by the output sequences of target models or systems. These model properties are usually already

known by the system or defined by the users before prediction, e.g., constraints by the physical world, or rules followed by the application domains (e.g., robotics). In practice, we can also mine the properties from the models' historical behaviors [109]. Actively learning these model properties helps build more robust deep neural networks.

Specification Language – Signal Temporal Logic: In order to enforce RNNs to learn the model properties, we first formalize properties using a machine-understandable specification language. For multivariate sequence prediction, capturing the relations between variables on the temporal domain is the most important task. Therefore, we apply STL [106] to formalize the model properties. STL is a very powerful formalism used to specify temporal properties of discrete and continuous signals. In this chapter, we target the properties of the outputs of RNN models, which are discrete-time signals. We first recall the syntax of an STL formula φ defined as follows,

$$\varphi ::= \mu \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \Diamond_{(a,b)}\varphi \mid \Box_{(a,b)}\,\varphi \mid \varphi \mathbf{U}_{(a,b)}\varphi.$$

We call μ a signal predicate, which is a formula in the form of $f(x) \geq 0$ with a signal variable $x \in \mathcal{X}$ and a function $f : \mathcal{X} \to \mathbb{R}$. The temporal operators $\Box$, $\Diamond$, and $\mathbf{U}$ denote "always", "eventually" and "until", respectively. The bounded interval (a, b) denotes the time interval of temporal operators. $\mathbf{U}$ can be expressed by $\Box$ and $\Diamond$, thus we only consider $\Box$ and $\Diamond$ in this work.

Model Properties and Formalization: Systems from different application domains have varied types of model properties. Focusing on the CPSs, we identify several critical types (not necessarily a complete list) of model properties for the key applications below. We give specific examples of properties under each type in Table 5.1.

- *Reasonable Range:* One of the most fundamental model properties is that the value of the sequence should always be within a reasonable range constrained by the system or physical world, such as the road capacity of vehicles, normal ambient temperature, etc. It is not trivial for RNNs to learn reasonable ranges since they could vary by variable, conditionally relate to another variable, and dynamically change over time.

- *Consecutive Changes:* For most applications in CPSs and other domains, the consecutive changes of the target model over a fixed period follow specific properties, such as pollution levels or traffic volume levels from one time period to the next are bounded.

Table 5.1: Examples of Model Properties and Their STL Formulas

Property Type	**Example**	**STL formula**
Reasonable Range	The traffic volume on a road can never exceed the road capacity.	$\Box_{[0,24]}(x_1 < \alpha_1) \wedge \cdots \wedge \Box_{[0,24]}(x_n < \alpha_n)$
Consecutive Changes	The number of people in a shopping mall should not increase or decrease more than 1000 in 10 min if exits number is less than 5.	$y < 5 \rightarrow \Box_{[0,10]}(\Delta x < 1000)$
Resource Constraint	The total energy distributed to all buildings should be less than e.	$\Box_{[0,24]}\mathsf{sum}(x_1, \ldots, x_n) < e$
Variable and Temporal Correlation	For two consecutive intersections on a one-way direction road, if there are 10 cars passing intersection A, then there should be at least 10 cars passing intersection B within the next 5 minutes.	$(x_1 > 10 \rightarrow \Diamond_{[0,5]}(x_2 > 10)) \wedge \cdots \wedge (x_n > 10 \rightarrow \Diamond_{[0,5]}(x_{n+1} > 10))$
Existence	There should be at least 1 patrol car around school every day.	$\Diamond_{[0,24]} x_1 \geq 1 \wedge \cdots \wedge \Diamond_{[0,24]} x_n \geq 1$
Unusual Cases	If there is a concert on Friday, the number of people in the nearby shopping mall will increase at least 200 within 2 hours.	$x_{\mathsf{Event}} = \mathsf{True} \wedge x_{\mathsf{Day}} = \mathsf{Fri} \rightarrow \Diamond_{[0,2]} \Delta x > 200$.

- *Resource Constraint:* The target models are often constrained by the resources, such as the available police resources to deal with an accident, or the maximum energy allocated by several locations. The resource constraints could also change over time in a real deployment, and are not necessarily the same as the training data. RNNs are highly likely to produce inaccurate or wrong outcomes without adapting the prediction results based on resource constraints.

- *Variable and Temporal Correlation:* There are correlations between different variables or locations over time, some of which are already known or easily discovered before training the learning model. These include the differences in air quality levels of adjacent locations, correlations between the air quality levels and traffic volume in the past hour, etc.

- *Existence:* Existence is a prevalent property in practice, but extremely difficult for RNNs to predict. It specifies the case that at least one of the values in the sequence (eventually) satisfies a specific property, e.g., traffic will be back to normal within 30 min after resolving an accident.

- *Unusual Cases:* The outputs of CPSs are highly affected by the environment and sensitive to uncertainties. For some unusual cases, there is a limited amount of data available in the training set. It is necessary to specify and teach the networks to learn the properties of these unusual cases (e.g., the influence of accidents or events on the population).

In Table 5.1, we present examples of these model properties and how to formalize them using STL. As we can see from the examples, most of the properties have temporal features over a given period,

e.g., $[0,2]$ indicates the next 2 hours from the checking point, $[0,24]$ indicates the next 24 hours (as checking every hour for the whole day). These properties describe the essential features of the systems with complex temporal dependency among multiple variables. Traditional RNNs have no mechanism to check or learn them explicitly.

5.2 Problem Formulation

With the model properties specified, we formally define the logic enforced learning problem. Let $\omega = \{\omega^1, \omega^2, \ldots, \omega^m\}$ denotes the target sequences of data with m variables over a finite discrete time domain $\mathbb{T}$ such that for the kth variable, $\omega^k[t] = x_t^k$ at any time $t \in \mathbb{T}$. Let $x_{[0,i]}^k$ be a prefix of sequence ω^k over the time domain $\{t_0, ..., t_i\} \subseteq \mathbb{T}$, and let $x_{[i+1,n]}^k$ be a suffix of sequence ω^k over the time domain $\{t_{i+1}, ..., t_n\} \subseteq \mathbb{T}$, where n denotes the total time instances, thus we denote the target sequence as $\omega^k = x_{[0,i]}^k x_{[i+1,n]}^k$. We have a deep learning prediction model f with parameter θ that predicts a suffix sequence with its prefix as inputs, i.e., $(\hat{x}_{[i+1,n]}^1, \hat{x}_{[i+1,n]}^2, \ldots, \hat{x}_{[i+1,n]}^m) = f((x_{[0,i]}^1, x_{[0,i]}^2, \ldots, x_{[0,i]}^m); \theta)$. We denote the predicted sequence as $\hat{\omega} = \{\hat{\omega}^1, \hat{\omega}^2, \ldots, \hat{\omega}^m\}$, where $\hat{\omega}^k = x_{[0,i]}^k \hat{x}_{[i+1,n]}^k$. Suppose the target sequence ω is drawn from a data distribution, i.e., $\omega \leftarrow \mathbf{w}$, and satisfies a set of properties, i.e., $\omega \vDash \varphi_1 \wedge \varphi_2 \wedge ... \wedge \varphi_\nu$, The goal is to find the model parameter θ that minimizes the distance (for a predefined distance metric $\mathcal{D}$) between the predicted sequence and the target sequence, and enforces the predicted sequence follows the same properties as well, i.e.,

$$\hat{\theta} = \arg\min_{\theta} \mathbb{E}_{\omega \leftarrow \mathbf{w}} [\mathcal{D}(\omega, \hat{\omega})]$$

$$s.t. \;\; \hat{\omega} \vDash \varphi_1 \wedge \varphi_2 \wedge ... \wedge \varphi_\nu$$

5.3 STLnet Framework

Our solution is STLnet which enforces multivariate RNNs to return results that follow the model properties of the system. In this section, we first introduce the construction of STLnet in the training phase and show how to enforce the results to guarantee the satisfaction in the testing phase. Then, we present the STL trace generator, which is the key component of the teacher network.

Following the idea of knowledge distillation [32], the STLNet framework is built with a teacher network and a student network (as shown in Figure 5.1). The main idea is that whenever the student network fails to predict a sequence that follows the model properties (detected by the STL trace

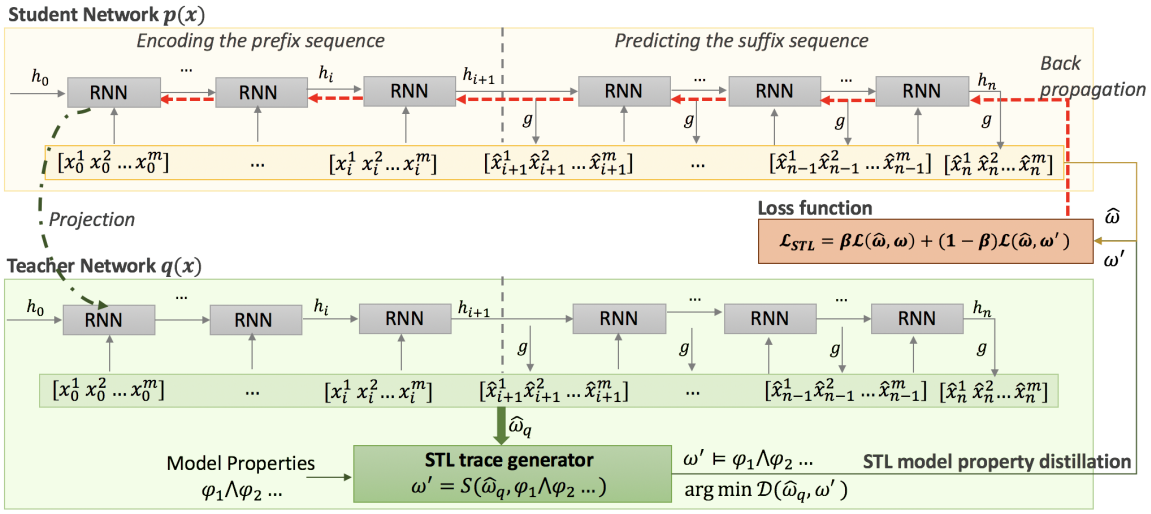


Figure 5.1: STLnet Framework

generator component), the teacher network generates a trace that is close to the trace returned by the student network and satisfies the model properties simultaneously. The student network then updates its parameters by learning from both the target trace and outcome of the teacher network.

In the *training* phase, our goal is to teach STLnet to learn from the "correct" traces, which include three major steps.

Step 1 - Student network construction: To start with, we build the basic student network, i.e., a general multivariate RNN f (e.g., LSTM, GRU, Bi-LSTM, etc.). It takes the past states as inputs and predict their future states in n time units, $(\hat{x}^1_{[i+1,n]}, \hat{x}^2_{[i+1,n]}, \ldots, \hat{x}^m_{[i+1,n]}) = f((x^1_{[0,i]}, x^2_{[0,i]}, \ldots, x^m_{[0,i]}); \theta)$. We denote the predicted sequence as $\hat{\omega} = \{\hat{\omega}^1, \hat{\omega}^2, \ldots, \hat{\omega}^m\}$, where $\hat{\omega}^k = x^k_{[0,i]} \hat{x}^k_{[i+1,n]}$ (i.e., the yellow box in Figure 5.1).

Step 2 - Teacher network construction: Next, we construct the teacher network $q(x)$ to generate a trace that satisfies the model properties $\varphi_1 \wedge \varphi_2 \wedge \cdots \wedge \varphi_\nu$ and has the shortest distance to the original prediction. We first formalize the model properties using STL. q is constructed by projecting p into a subspace constrained by the properties. Different than the structure of the student network p, q has an STL trace generator. The STL trace generator first checks if this trace follows the properties, if yes, then output the trace $\omega' = \hat{\omega}$. If not, it generates a new trace ω', which first follows the properties, $\omega' \vDash \varphi_1 \wedge \varphi_2 \cdots \wedge \varphi_m$ and secondly, it is the closest trace to the original predicted trace $\hat{\omega}$. Here we use L_1 distance to measure the distance between two traces, i.e., the total amount of changes. We present the details of STL trace generator in Section 5.4.

Step 3 - Back propagation with loss $\mathcal{L}_{\mathbf{STL}}$: The loss function is constructed by two parts to guide the student network $p(x)$ to balance between emulating the teacher's output and predicting the target trace. The target trace is ω, thus the first part of the loss is $\mathcal{L}(\hat{\omega}, \omega)$, where $\mathcal{L}$ calculates the L-2 distance between two traces. The second part of the loss function is defined by the L-2 distance between the predicted trace $\hat{\omega}$ and teacher's output ω', i.e., $\mathcal{L}(\hat{\omega}, \omega')$. Thus, the student network is back propagated using the loss function as,

$$\mathcal{L}_{\text{STL}} = \beta\mathcal{L}(\hat{\omega}, \omega) + (1 - \beta)\mathcal{L}(\hat{\omega}, \omega') \tag{5.1}$$

The network is trained iteratively by repeating Steps 2 and 3 until convergence.

Similar to other distilled networks [33], in the *testing* phase, we can use either the distilled student network p or the teacher network q after a final projection. Our results show that both models substantially improve over the base network that is trained without STL specified properties. In practice, q can guarantee the satisfaction of model properties while p is more lightweight and efficient. We compare the performance of p and q with baseline network extensively in the evaluation.

5.4 STL Trace Generator

In this subsection, we introduce the algorithms of the key component of the teacher network, i.e., the STL trace generator. It is easy to check if a trace satisfies a specific property, however, to train the network, STLnet also needs to obtain the closest satisfying trace when the prediction results violate a property. It is a very difficult and time-consuming task.

In this chapter, tailoring to the deep learning process, we create a STL trace generator. The key idea is to obtain a small Disjunctive Normal Form (DNF, a canonical normal form of a logical formula consisting of a disjunction of conjunctions) set representing the possible satisfaction ranges of each time on the trace before the optimization of the Deep Learning model. Then, we obtain the closest satisfaction trace of each instance in the training and testing phase. A simplified example of STL trace generator with a single variable is demonstrated in Figure 5.2.

Converting STL to DNF We first convert the STL formula into DNF and calculate the satisfaction range for each time unit on the sequence (left part of Figure 5.2). A nice property of a DNF representation is that for a trace to satisfy the requirement, it is a sufficient and necessary condition

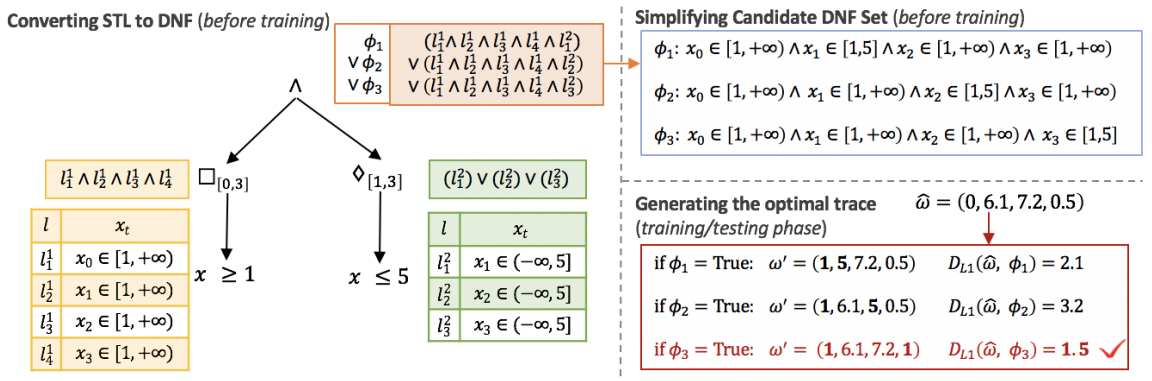


Figure 5.2: An example of STL trace generator ($\varphi = \square_{[0,3]} x \geq 1 \wedge \Diamond_{[1,3]} x \leq 5$)

for it to match some clause ϕ inside φ. Therefore, we can check the properties in a straightforward manner by comparing the distance of the trace to each of the clauses in formula φ.

Proposition 5.1 (STL formula in DNF representation)**.** *Every STL φ can be represented in the DNF formula $\xi(\varphi)$, where $\xi(\varphi)$ is a formula that includes several clauses ϕ_k that are connected with the disjunction operator, and the length of ϕ_k is denoted by $|\phi_k|$. Each clause ϕ_k can be further represented by several Boolean variables l_i that are connected with the conjunction operator. Finally, each Boolean variable l_i is the satisfaction range of a specific parameter.*

$$\begin{aligned} \xi(\varphi) &= \phi_1 \vee \phi_2 \vee \ldots \vee \phi_K \\ \phi_k &= l_1^{(k)} \wedge l_2^{(k)} \wedge .. \wedge l_{|\phi_k|}^{(k)} \quad \forall k \in \{1, 2..K\} \\ l_i^{(k)} &= \{x_t^j \mid f(x_t^j) \geq 0\} \text{ where } (t \in T), \forall i \in \{1, 2..|\phi_k|\} \end{aligned} \tag{5.2}$$

Proof. We prove Proposition 4.1 by induction. We use induction on the top-layer operator:

- A single μ operator can be represented by a single l clause, where $f(x_0) \geq 0$.
- If the low layer operators can be represented by DNF formula, the result of $\neg$, $\wedge$, and $\vee$ operators can also be represented as DNF formula by the De Morgen rule.
- The always operator $\square_{(a,b)}\phi$ can be decomposed as multiple $\wedge$ operator on the time period (a, b). Given the DNF φ and a specific time $t \in (a, b)$, the actual DNF should be φ with an additive time shift t on every time operator in φ. Then the STL formula is equivalent to a DNF built by applying the De Morgen rule on the DNFs with every $t \in (a, b)$.

- The eventually operator $\Diamond_{(a,b)}\phi$ can be decomposed as multiple $\vee$ operator on the time period (a,b). Given the DNF φ and a specific time $t \in (a,b)$, the actual DNF should be φ with an additive time shift t on every time operator in φ. Then the STL formula is equivalent to a DNF built by connecting the DNFs for every $t \in (a,b)$ with $\vee$.

- The until operator $\mathbf{U}_{(a,b)}$ by the STL definition can be represented with $\Box$ and $\Diamond$ operators. Therefore it can also be represented by a DNF.

By induction, we have Proposition 5.1 proved. $\Box$

Algorithm 5.1 shows how to construct the DNF representation of the STL formula. The algorithm follows a top-down recursive manner. For every operator and its corresponding sub-tree, we first calculate the DNF formula of its children sub-trees, and then combine them with the operator. Specifically, the negation operator causes the DNF formula to become a CNF. Therefore, here we use De Morgan rule to sink the negation operator to the bottom level. Each clause of the DNF formula is guaranteed to have no duplicate variables. To be noted, if the set returned is an empty set, we know that the input requirement is unsatisfiable and we don't progress to the next steps. The computation time of Algorithm 5.1 is relevant to the number of predicates in the STL formula. We only execute it once in the pre-processing step before the training phase, thus it tailors to deep learning processes efficiently.

Simplifying Candidate DNF Set: In order to further obtain a smaller manageable DNF set, we reduce the size of the DNF set by finding the overlaps between the clauses. We first define the distance between a trace and a clause in DNF. (Note that we use $L1$ distance in the definition, which can be extended to any Lp distance measure.)

Definition 5.1 ($L1$-Distance of a trace to a clause)**.** *Let clause $\phi = l_1 \wedge l_2 \wedge .. \wedge l_m$. The distance between a trace ω and clause ϕ is defined as*

$$\begin{aligned} &D_{L1}(\omega,\phi) = \min_{\omega'} \sum_{t=1}^{T} |\omega'_t - \omega_t|, \\ &\textit{where} \quad \omega' \vDash l_i \qquad \forall i \in \{1,2..m\} \end{aligned} \tag{5.3}$$

Algorithm 5.1: Converting STL to DNF with Calculation of Satisfaction Range

```
Function CalDNF(φ, t, sgn):
    Input: STL Formula φ, time t, sign sgn
    Output: DNF Set ξ representing the satisfaction range
    begin
        if sgn = False then
            switch φ do
                Case μ
                    return {x_t^j | f(x_t^j) < 0};
                Case ¬φ
                    CalDNF(φ, t, True);
                Case φ1 ∧ φ2
                    return CalDNF(¬φ1 ∨ ¬φ2, t, True);
                Case φ1 ∨ φ2
                    return CalDNF(¬φ1 ∧ ¬φ2, t, True);
                Case □_T φ
                    return CalDNF (◊_T ¬φ, t, True);
                Case ◊_T φ
                    return CalDNF (□_T ¬φ, t, True);
            end
        end
        else
            switch φ do
                Case μ
                    return {x_t^j | f(x_t^j) ≥ 0};
                Case ¬φ
                    return CalDNF(φ, t, False);
                Case φ1 ∧ φ2
                    ξ ← ∅;
                    for φ1 ∈ CalDNF(φ1, t, sgn) do
                        for φ2 ∈ CalDNF(φ2, t, sgn) do
                            ξ ← ξ ∨ (φ1 ∧ φ2);
                        end
                    end
                    return ξ;
                Case φ1 ∨ φ2
                    ξ1 ← CalDNF(φ1, t, sgn) ;
                    ξ2 ← CalDNF(φ2, t, sgn) ;
                    return ξ1 ∨ ξ2;
                Case □_T φ
                    ξ ← {True};
                    for t ∈ T do
                        ξ1 ← CalDNF(φ1, t, sgn) ;
                        ξ2 ← ∅;
                        for φ1 ∈ ξ do
                            for φ2 ∈ ξ1 do
                                ξ2 ← ξ2 ∨ (φ1 ∧ φ2);
                            end
                        end
                        ξ ← ξ2;
                    end
                    return ξ;
                Case ◊_T φ
                    ξ ← ∅;
                    for t ∈ T do
                        ξ ← ξ ∨ CalDNF(φ, t, sgn);
                    end
                    return ξ;
            end
        end
    end
```

The return value is a non-negative real number. If a variable satisfies a constrain in a clause, the term will be evaluated to 0; Otherwise, it will return the minimal distance over all the items in the satisfaction of l_i (not necessary to be 1).

Proposition 5.2 (The order of set)**.** *For two clauses ϕ_i and ϕ_j in a DNF ξ, if $\forall \omega \vDash (\phi_i), \omega \vDash \phi_j$, and $\phi_i \subseteq \phi_j$, then we have $D_{L1}(\omega, \phi_i) \leq D_{L1}(\omega, \phi_j)$.*

Proof. Prove by contradiction. Assume $D_{L1}(\omega, \phi_i) > D_{L1}(\omega, \phi_j)$. Let ω' denotes the trace with minimal distance to ω in ϕ_j, that is, $\omega' = \arg\min_{\omega'' \vDash \phi_j} D_{L1}(\omega, \omega'')$. As $\phi_i \subseteq \phi_j$, we have $\omega' \vDash \phi_i$. Therefore, $D_{L1}(\omega, \phi_i) = \min_{\omega'' \vDash \phi_i} D_{L1}(\omega, \omega'') \leq D_{L1}(\omega, \omega')$, which clearly contradicts the assumption. Therefore, $D_{L1}(\omega, \phi_i) \leq D_{L1}(\omega, \phi_j)$. □

If the satisfaction set of one clause is the subset of the satisfaction set of another clause, the first clause is unnecessary, as stated in Proposition 5.2. Therefore, we provide a pairwise comparison between all clauses, and we can remove some of the clauses and obtain a smaller set of DNF before training.

Generating the optimal trace To satisfy a DNF representation, at least one of the clauses ϕ_k needs to be satisfied. Therefore, the best ω' is found by a specific k, as formally stated in Proposition 5.3.

Proposition 5.3 (Shortest distance of a trace to the DNF formula)**.** *Let $\hat{\omega}$ be the trace that satisfy the DNF formula $\varphi = \phi_1 \vee \phi_2 \vee ... \vee \phi_K$ that has minimal distance to the input trace ω, then we have*

$$\hat{k} = \arg\min_k D_{L1}(\omega, \phi_k) \tag{5.4}$$

and $\hat{\omega}$ is the trace that minimizes $D_{L1}(\omega, \phi_{\hat{k}})$ by $D_{L1}(\omega, \hat{\omega}) = D_{L1}(\omega, \phi_{\hat{k}})$.

Proof. We prove the proposition by contradiction. Assume $\hat{\omega} \vDash \varphi$, by the definition of DNF formula, we have $\exists k : \hat{\omega} \vDash \phi_k$. Suppose $\hat{k}$ is one of choices that $\hat{\omega} \vDash \phi_{\hat{k}}$.

If $\hat{k} \neq \arg\min_k D_{L1}(\omega, \phi_k)$, then there exists another k' that $D_{L1}(\omega, \phi_{k'}) < D_{L1}(\omega, \phi_{\hat{k}})$. By the definition of $D_{L1}(\omega, \phi_{k'})$, there exists a ω' that $D_{L1}(\omega, \omega') = D_{L1}(\omega, \phi_{k'}) < D_{L1}(\omega, \phi_{\hat{k}}) = D_{L1}(\omega, \hat{\omega})$. We also have $\omega' \vDash \phi_{k'}$, which indicates $\omega' \vDash \varphi$. Then ω' is closer to ω and also satisfies φ, which contradicts the assumption.

If $\hat{\omega}$ doesn't minimize $D_{L1}(\omega, \phi_{\hat{k}})$, then there exists another $\omega' \vDash \phi_{\hat{k}}$ that $D_{L1}(\omega, \omega') = D_{L1}(\omega, \phi_{\hat{k}}) < D_{L1}(\omega, \hat{\omega})$. Then ω' is closer to ω and also satisfies φ, which contradicts the assumption. □

For each clause in the DNF set, we calculate the distance between the trace to be optimized with the clause. The distance can be then calculated by a summation over the distance of the satisfaction of all the Boolean variables. After the distance calculation, we return the optimal trace with the minimal distance as our generated target trace. The returned trace is therefore guaranteed to satisfy the requirement. In the example given in Figure 5.2 (right side), if the trace predicted by the student network is $\hat{\omega} = (0, 6.1, 7.2, 0.5)$, then, the optimal new trace generated by the teacher network is $\omega' = (\mathbf{1}, 6.1, 7.2, \mathbf{1})$, which has the shortest distance to $\hat{\omega}$ and satisfies its model property.

5.5 Evaluation

We evaluate the performance of STLnet from two aspects: the capability of learning different types of model properties and the performance in learning from a city dataset with multiple variables.

In all experiments, we evaluate the performance using three *metrics*, i.e., Root Mean Square Error (RMSE), property satisfaction rate, and average STL robustness value ρ. RMSE measures the accuracy of the prediction, property satisfaction rate shows the percentage of the predicted sequence that satisfies the property, and STL robustness value ρ shows the degree of satisfaction (we refer paper [106] or our supplementary materials for the definition of quantitative semantics). Briefly, if $\rho \leq 0$, the property is violated. The smaller ρ is, the more the property is violated.

To evaluate the performance of STLnet, we applied it to two networks, i.e., an LSTM network [107] and a transformer network [108] for multivariate sequential prediction. For each model, we compare the experimental results among three networks, i.e., general model, model with STLnet testing with the student network (p), and model with STLnet testing with the teacher network (q). Applying STLnet-q can guarantee the satisfaction of all model properties, so we also present the results of STLnet-p here to show the improvement we achieved through training. To be noted, STLnet is general and can apply to all RNNs. We use LSTM and Transformer networks as examples. The experiments are evaluated on a server machine with 20 CPUs, each core is 2.2GHz, and 4 Nvidia GeForce RTX 2080Ti GPUs. The operating system is Centos 7.

5.5.1 Learning Model Properties

The *goal* of the first set of experiments is to show that STLnet is general and robust to different types of model properties and improves the satisfaction rate significantly. To start with, we synthesize six sets of data that satisfy six types of model properties, respectively.

- *Resource constraint:*

  To synthesize the data with the model property of resource constraint, we use a piecewise constant function to generate n_d instances, each following:

$$x_1(t) = x_2(t) = \begin{cases} 1.0 - \sigma(t) & t < d \\ 1.005 + \sigma(t) & t \geq d. \end{cases} \tag{5.5}$$

  where $\sigma(t)$ is a small Gaussian noise, and d is pick randomly between 10 to 14. The function follows model property φ_1, which is used in STLnet to enhance learning,

$$\varphi_1 = \square_{[0,8]}\neg(x_1 > 1) \wedge \square_{[14,19]}(x_1 > 1) \wedge \square_{[0,8]}\neg(x_2 > 1) \wedge \square_{[14,19]}(x_2 > 1). \tag{5.6}$$

- *Consecutive change:*

  To synthesize the data with the model property of consecutive change, we use a monotonically decreasing function to generate n_d sequences, each following:

$$\begin{aligned} x_1(t) &= x_1(t-1) - \min(100, 0.2x_1(t-1)) \\ x_2(t) &= x_2(t-1) - \min(100, 0.2x_2(t-1)). \end{aligned} \tag{5.7}$$

  We pick the original value $x_1(0)$ and $x_2(0)$ uniformly between the range $[0, 1000)$. The function follows model property φ_2, which is used in STLnet to enhance learning,

$$\varphi_2 = \square_{[0,19]}(\neg(\Delta x_1 > 100) \wedge \neg(\Delta x_2 > 100)). \tag{5.8}$$

- *Variable and Temporal Correlation:*

To synthesize the data with the model property of variable and temporal correlation, we generate to generate n_d sequences. Each sequence consists only 0 and 1, but keep not any group of 4 consecutive numbers to be the same. That is,

$$x_1(t) = \begin{cases} 0 & \text{If } x_1(t-1) = 1 \wedge x_1(t-2) = 1 \wedge x_1(t-3) = 1 \\ 1 & \text{If } x_1(t-1) = 0 \wedge x_1(t-2) = 0 \wedge x_1(t-3) = 0 \\ \text{Bernoulli}(0.5) & \text{Otherwise.} \end{cases} \tag{5.9}$$

The function follows model property φ_3, which is used in STLnet to enhance learning,

$$\varphi_3 = \square_{[0,5]} \left(\Diamond_{[0,4]}(x_1 > 0) \wedge \Diamond_{[0,4]}(\neg(x_1 > 0)) \right). \tag{5.10}$$

- *Reasonable range:*

To synthesize the data with the model property of reasonable range, we use a periodic function to generate n_d sequences, each following:

$$\begin{aligned} x_1(t) &= \sin(at + b) \\ x_2(t) &= \cos(at + b). \end{aligned} \tag{5.11}$$

Where a is uniformly picked from $[0.77, 1.03)$, and b is uniformly picked from $[0, 0.5)$. The function follows model property φ_4, which is used in STLnet to enhance learning,

$$\varphi_4 = \square_{[0,19]}(x_1 > -1.0 \wedge \neg(x_1 > 1.0) \wedge x_2 > -1.0 \wedge \neg(x_2 > 1.0)). \tag{5.12}$$

- *Existence:*

To synthesize the data, we generate n_d instances of 0 and 1. In each sequence make sure that for both x_1 and x_2 it equals 1 at a single t and equals 0 at other time. The function follows model property φ_5, which is used in STLnet to enhance learning,

$$\varphi_5 = \Diamond[0, 19](x_1 > 0.99) \wedge \Diamond[0, 19](x_2 > 0.99). \tag{5.13}$$

- *Unusual cases:*

To synthesize the data with the model property of unusual cases, we generate n_d instances following:

$$x_1(t) = \begin{cases} 1000 & t = t_d \\ 0 & \text{otherwise.} \end{cases} \tag{5.14}$$

and

$$x_2(t) = \begin{cases} 10 & \exists t_i \in [1,9], x_1(t-t_i) > 0 \\ \sigma(t) & \text{otherwise.} \end{cases} \tag{5.15}$$

where d is pick randomly between 0 to 4, and $\sigma(t)$ is a small Gaussian noise.

The function follows model property φ_6, which is used in STLnet to enhance learning,

$$\varphi_6 = \square_{[0,4]}(x_1 > 500 \vee \square_{[1,9]} x_2 > 9). \tag{5.16}$$

The results shown in Table 5.2 and Table 5.3 are obtained from 25 runs. From the results, we can see that: (1) STLnet significantly improves the model property satisfaction rate for both LSTM and Transformer. For all the property, both satisfaction rate and violation degree are improved by STLnet. For example, property φ_5 and φ_4, the satisfaction rates of basic LSTM are only 0.84% and 56.68% with very high violation degrees (-463.534 and -36.884), while STLnet-p achieves 75.64% and 83.09% satisfaction rate with violation degree dropping to -19.842 and -3.906, respectively. STLnet-q can guarantee the satisfaction of all the model properties. (2) For φ_1 to φ_6, STNnet not only improves the satisfaction rate, but also decreases the RMSE value. In general, STL-q can further improve the accuracy. For example, property φ_6 (property type of unusual cases), RNNs are not able to learn the unusual cases which have a small portion of instances in the training data and lead to a low satisfaction rate. While STLnet guides the predicted trace follow the property (in both training and testing), it also decreases RMSE. It indicates that learning model properties can support RNNs to build a more accurate model. Overall, the results prove the effectiveness and generalizability of STLnet dealing with different types of model properties.

5.5.2 Multivariate Air Quality Prediction with Model Properties

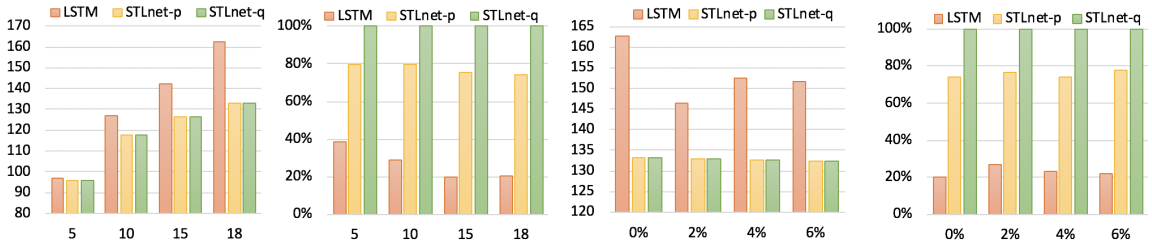
The *goal* of the second set of experiments is to show how STLnet improves the accuracy and robustness of RNNs in a real-world CPS application, especially in cases of noisy/missing sensing data, and long term prediction. We apply STLnet to train RNN models with air quality datasets.

Table 5.2: Comparison of Accuracy and Property Satisfaction among LSTM, STLnet-p and STLnet-q

	LSTM			**LSTM STLnet-p**			**LSTM STLnet-q**		
	RMSE	Sat Rate	Violateρ	RMSE	Sat Rate	Violateρ	RMSE	Sat Rate	Violateρ
φ_1	0.026	92.00%	-0.298	0.025	98.34%	-0.014	0.025	100.00%	0
φ_2	94.304	75.61%	-117.982	90.016	97.78%	-1.603	90.160	100.00%	0
φ_3	4.214	75.47%	-1.589	4.209	87.69%	-0.606	4.209	100.00%	0
φ_4	0.309	56.68%	-36.884	0.230	83.09%	-3.906	0.229	100.00%	0
φ_5	2.188	0.84%	-463.534	1.151	75.64%	-19.842	1.162	100.00%	0
φ_6	8.603	59.54%	-282.403	8.532	61.85%	-282.403	7.122	100.00%	0

Table 5.3: Comparison of Accuracy and Property Satisfaction among Transformer Model, STLnet-p and STLnet-q

	Transformer			**Transformer STLnet-p**			**Transformer STLnet-q**		
	RMSE	Sat Rate	Violateρ	RMSE	Sat Rate	Violateρ	RMSE	Sat Rate	Violateρ
φ_1	0.045	27.76%	-18.808	0.031	89.48%	-1.835	0.031	100.00%	0
φ_2	105.211	49.44%	-109.282	111.688	76.08%	-18.874	111.655	100.00%	0
φ_3	4.340	52.96%	-3.855	4.339	60.70%	-2.596	4.339	100.00%	0
φ_4	0.124	0.36%	-38.893	0.135	51.00%	-5.101	0.135	100.00%	0
φ_5	2.196	8.88%	-31.172	1.805	50.20%	-4.612	1.804	100.00%	0
φ_6	8.156	20.08%	-301.175	8.326	20.32%	-307.165	2.657	100.00%	0



(a) RMSE-prediction lengths (b) Satisfaction Rate - prediction lengths (c) RMSE -missing data % (d) Satisfaction Rate - missing data %

Figure 5.3: Comparison of RMSE and Satisfaction Rate among LSTM, STLnet-p and STLnet-q

The dataset includes 1.3 million instances of 6 pollutants (i.e., PM2.5, PM10, CO, SO_2, NO_2, O_3) collected from 130 locations in Beijing every hour between 5/1/2014 and 4/30/2015 [103]. To build the LSTM network, we regard one pollutant from one location as one variable, and concatenate all variables from the same time unit. Next, we specify important model properties, including reasonable ranges, consecutive changes, correlations between different pollutants, and between different locations, etc.

The results of the comparison are presented in Figure 5.3. From the results, we can see that, (1) STLnet improves both property satisfaction rate (i.e., from 20% to over 70% on average) and RMSE (i.e., from about 150 to 130 on average). STLnet-q outperforms STLnet-p regarding the satisfaction rate and achieves a similar RMSE as STLnet-p. (2) Figure 5.3 (a) and (b) compare the performance with different time lengths of prediction. When predicting future 5-time units, three networks have a very similar RMSE value. With the prediction length increasing, the RMSE value of LSTM increases

greatly. However, with STLnet, the prediction accuracy is improved, e.g., when $l = 18$, RMSE drops from 162 to 132 (18.5%). (3) Datasets with missing data affect the learning performance. We test the model performance with different percentages of missing data. The results show that STLnet is able to improve model accuracy by as much as 14% and property satisfaction rate by 3 to 4 times (p and q, respectively). Overall, the results indicate the effectiveness and robustness of STLnet in a real-world application. It also can support RNN models to perform long-term prediction with missing data.

5.6 Summary

In order to guide Multivariate RNNs to follow the model properties of the system and produce a more robust model for improved future predictions, we build STLnet, a new temporal logic-based learning framework. The experimental results show that STLnet not only improves the accuracy of predictions, but importantly also guarantees the satisfaction of model properties. The promising results also indicate that considering model properties is very important for building deep learning models for complex systems, and formal logic can be an effective way to enhance the robustness of the deep learning models.

The approach created in this chapter can be broadly applied to the tasks of sequence prediction using RNN models in application domains such as smart cities, smart health, and other CPS-IoT systems. In these systems prediction results are usually used to support monitoring and decision making processes. The goal is to improve the prediction accuracy and, more importantly, guarantee the satisfaction of critical properties. We envision that relevant systems and decision-makers will benefit from this work. In this way smart cities and smart health systems can improve safety and performance, thereby improving daily life and health for people. Failure of the system (i.e., the model produces wrong prediction results) could affect the decisions made based on the results. In practice, even if the prediction results are somewhat inaccurate (e.g., high RMSE), STLnet can still guarantee the satisfaction of key properties.

Chapter 6

Runtime Verification

Integrated Cyber-Physical Systems, such as smart cities, are emerging around the world. Examples include Chicago's Array of Things project [110], IBM's Rio de Janeiro Operations Center [96] and Cisco's Smart+Connected Operations Center [97], just to name a few. They utilize a vast amount of data and smart services to enhance the safety, efficiency, and performance of city operations [105]. In order to provide real-time services or safety protection, all services and city controllers have to monitor city's safety and performance and thus take actions accordingly in real time. For example, governments can use real-time traffic flow data to reduce traffic congestion and manage public transportation; city managers can optimize the energy usage via smart buildings and infrastructures based on the real-time energy demand; law enforcement units can benefit from big data generated from various sensors in the city to keep the citizens safe. [111]. City controllers, such as discussed in Chapter 4, CityGuard predicts and monitors the changing of city states caused by the services to detect and prevent potential safety violations caused by independently-developed services. The safety violations bring serious consequences to the city environment and citizen's safety. Therefore, there is a need for monitoring city states in real-time to ensure safety and performance requirements [93]. If a requirement violation is detected by the monitor, the city operators and smart service providers can take actions to change the states, such as improving traffic performance, rejecting unsafe actions, sending alarms to police, etc. The key **challenges** of developing such a monitor include how to use an expressive, machine-understandable language to specify smart city requirements, and how to efficiently monitor requirements that may involve multiple sensor data streams (e.g., some requirements are concerned with thousands of sensors in a smart city).

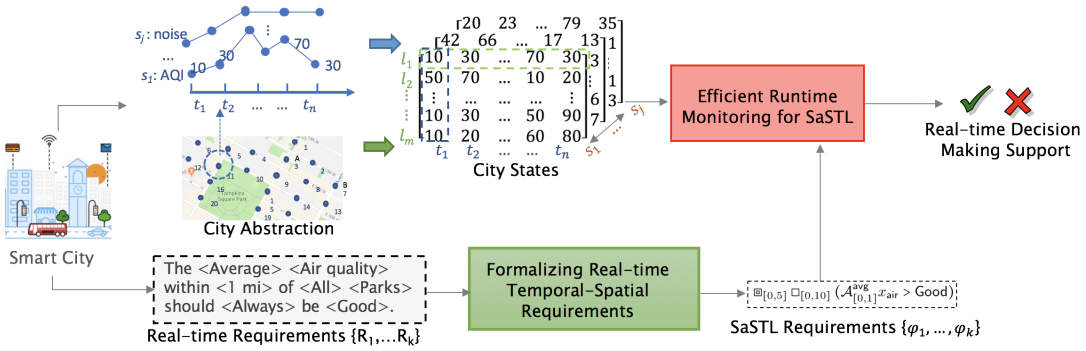


Figure 6.1: A framework for runtime monitoring of real-time city requirements

Previous works [112, 113, 98, 114] have proposed solutions to monitor smart cities using formal specification languages and their monitoring machinery. In Chapter 4, we introduced CityResolver, [38] which uses Signal Temporal Logic (STL) [50] to support the specification-based monitoring of safety and performance requirements of smart cities. However, STL is not expressive enough to specify smart city requirements concerning *spatial* information such as *"the average noise level within 1 km of all elementary schools should always be less than 50 dB"*. There are some existing spatial extensions of STL (e.g., SSTL [49], SpaTeL [114] and STREL [55, 56], see [115] for a recent tutorial), which can express requirements such as *"there should be no traffic congestion on all the roads in the northeast direction"*. But they are not expressive enough to specify requirements like *"there should be no traffic congestion on all the roads on average"*, or *"on 90% of the roads"*, which require the aggregation and counting of signals in the spatial domain. To tackle these challenges and limitations, we develop a novel Spatial Aggregation Signal Temporal Logic (SaSTL), which extends STL with two new logical operators for expressing spatial aggregation and spatial counting characteristics which we demonstrate are commonly found in real city requirements. More specifically, this chapter has the following major **contributions**:

- To the best of our knowledge, this is the first work studying and annotating over 1,000 real smart city requirements across different service domains to identify the gap of expressing smart city requirements with existing formal specification languages. As a result, we found that aggregation and counting signals in the spatial domain (e.g., for representing sensor signals distributed spatially in a smart city) are extremely important for specifying and monitoring city requirements.

- Drawing on the insights from our requirements study, we develop a new specification language SaSTL, which extends STL with a *spatial aggregation* operator and a *spatial counting* operator.

SaSTL can be used to specify Point of Interests (PoIs), the physical distance, spatial relations of the PoIs and sensors, aggregation of signals over locations, degree/percentage of satisfaction and the temporal elements in a very flexible spatial-temporal scale. We define Boolean and quantitative semantics with theoretical proofs.

- We compare SaSTL with some existing specification languages and show that SaSTL has a much higher coverage expressiveness (95%) than STL (18.4%), SSTL (43.1%) or STREL (43.1%) over 1,000 real city requirements.

- We develop novel and efficient monitoring algorithms for SaSTL. In particular, we present two new methods to speed up the monitoring performance: (i) dynamically prioritizing the monitoring based on cost functions assigned to nodes of the syntax tree, and (ii) parallelizing the monitoring of spatial operators among multiple locations and/or sensors. We show that both methods improve the time complexity of SaSTL monitoring algorithms.

- We evaluate the SaSTL monitor by applying it to monitoring real city data collected from Chicago and Aarhus. The results show that SaSTL monitor has the potential to help identify safety violations and support the city managers and citizens to make decisions. We also evaluate the SaSTL monitor on a third case study of conflict detection and resolution among smart services in simulated New York City with large-scale real sensing data (e.g., up to 10,000 sensors used in one requirement). Results of our simulated experiments show that SaSTL monitor can help improve the city's performance (e.g., 21.1% on the environment and 16.6% on public safety), with a significant reduction of computation time compared with previous approaches.

- We develop a SaSTL monitoring tool that can support decision making of different stakeholders in smart cities. The tool allows users (e.g., city decision maker, citizens) without any formal method background to specify city requirements and monitor city performance easily.

6.1 Overview

We envision that a monitoring framework would operate in a smart city's central control center (e.g., IBM's Rio de Janeiro Operations Center [96] or Cisco's Smart+Connected Operations Center [97]) where sensor data about city states across various locations are available in real time. Figure 6.1 shows an overview of our SaSTL runtime monitoring framework for smart cities. The framework would monitor city states and check them against a set of smart city requirements at runtime. The

monitoring results would be presented to city managers to support decision making. The framework makes abstractions of city states in the following way. The framework formalizes a set of smart city requirements (See Section 6.2) to some machine checkable SaSTL formulas (See Section 6.3). Different data streams (e.g. CO emission, noise level) over temporal and spatial domains can be viewed as a 3-dimensional matrix. For any signal s_j in signal domain S, each row is a time-series data at one location and each column is a set of data streams from all locations at one time. Next, the efficient real-time monitoring for SaSTL verifies the states with the requirements and outputs the Boolean satisfaction to the decision makers, who would take actions to resolve the violation. To support decision making in real time, we improve the efficiency of the monitoring algorithm in Section 6.4. We implement SaSTL runtime monitoring tool following this framework for city experts without any formal methods background (see Section 6.5). We describe more details of the framework in the following sections.

6.2 Analysis of Real City Requirements

To better understand real city requirements, we conduct a requirement study. We collect and statistically analyze 1000 quantitatively specified city requirements (e.g., standards, regulations, city codes, and laws) across different application domains, including energy, environment, transportation, emergency, and public safety from over 70 cities (e.g. New York City, San Francisco, Chicago, Washington D.C., Beijing, etc.) around the world. Some examples of these city requirements are highlighted in Table 6.1. We identify key required features to have in a specification language and its associated use in a city runtime monitor. The summarized statistical results of the study and key elements we identified (i.e., temporal, spatial, aggregation, entity, comparison, and condition) are shown in Table 6.2.

Temporal: Most of the requirements include a variety of temporal constraints, e.g. a static deadline, a dynamic deadline, or time intervals. In many cases (65.7%), the temporal information is not explicitly written in the requirement, which usually means it should be "always" satisfied. In addition, city requirements are highly real-time driven. In over 80% requirements, cities are required to detect requirement violations at runtime. Usually, it is a deadline like "within one minute", or time interval like "between the hours of 7:00 a.m. and 10 a.m.", "after 9:00 p.m." or "weekdays". It indicates a high demand for runtime monitoring.

Spatial: A requirement usually specifies its spatial range explicitly using the Points of Interest (PoIs)

Table 6.1: Examples of city requirements from different domains

(Key elements: temporal, spatial, aggregation, entity, condition, comparison.)

Domain	Example
Transportation	Limits vehicle idling to one minute adjacent to any school, pre-K to 12th grade, public or private, in the City of New York [116].
	The engine, power and exhaust mechanism of each motor vehicle shall be equipped, adjusted and operated to prevent the escape of a trail of visible fumes or smoke for more than ten (10) consecutive seconds [117].
	Prohibit sight-seeing buses from using all bus lanes between the hours of 7:00 a.m. and 10:00 a.m. on weekdays [118].
Energy	Operate the system to maintain zone temperatures down to 55°F or up to 85°F [119].
	The total leakage shall be less than or equal to 4 cubic feet per minute per 100 square feet of conditioned floor area [120].
Environment	It shall be unlawful for any person, between the hours of 8:00 p.m. of any day and 7:00 a.m. of the following day to erect, construct, demolish, excavate for, alter or repair any building or structure if the noise level created thereby is in excess of the ambient noise level by 5 dB at the nearest property plane, unless a special permit therefor has been applied for and granted by the Director of Public Works or the Director of Building Inspection [121].
	LA Sec 111.03 minimum ambient noise level table: ZONE M2 and M3 – DAY: 65 dB(A) NIGHT: 65 dB(A) [122].
	The total amount of HCHO emission should be less than 0.1mg per m^3 within an hour, and the total amount of PM10 emission should be less than 0.15 mg per m^3 within 24 hours [123].
Emergency	NYC Authorized emergency vehicles may disregard 4 primary rules regarding traffic [124].
	At least one ambulance should be equipped per 30,000 population (counted by area) to obtain the shortest radius and fastest response time [125].
Public Safety	Security staff shall visit at least once per week in public schools [126].

(80.1%), such as "park", "xx school", along with a distance range (65%). One requirement usually points to a set of places (e.g. all the schools). Therefore, it is very important for a formal language to be able to specify the spatial elements across many locations within the formula, rather than one formula for each location.

We also found that the city requirements specify a very large spatial scale. Different from the requirements of many other CPS, requirements from smart cities are highly spatial-specific and usually involve a very large number of locations/sensors. For example, the first requirement in Table 6.1 specifies a vehicle idling time "adjacent to any school, pre-K to 12th grade in the City of New York". There are about 2000 pre-K to 12th schools, even counting 20 street segments nearby each school, there are 40,000 data streams to be monitored synchronously. An efficient monitoring is highly demanded.

Aggregation: In 51.9% cases, requirements are specified on the aggregated signal over an area, such

Table 6.2: Key elements of city requirements and statistical results from 1000 real city requirements

Element	Form	Number	Example
Temporal	Dynamic Deadline	77	limit ... to one minute
	Static Deadline	98	at least once a week
	Interval	168	from 8am to 10am; within 24 hours;
	Default	657	The noise (always) should not exceed 50dB.
Spatial	**PoIs/Tags**	**801**	school area; all parks;
	Distance	**650**	Nearby
	Default	154	(everywhere) ; (all) locations
Aggregation	**Count, Sum**	**256**	in total; x out of N locations; %;
	Average	**196**	per m^2;
	Max, Min	**67**	highest/lowest value
Entity	Subject	1000	air quality; Buses;
Comparison	Value comparison	836	More than, less than
	Boolean	388	Street is blocked; should
	Not	456	It is unlawful/prohibited...
Condition	Until	24	keep... until the street is not blocked.
	If/Except	44	If rainy, the speed limit...

as, “the total amount”, “average...per 100 square feet”, “up to four vending vehicles in any given city block”, “at least 20% of travelers from all entrances should ...”, etc. The same set of data streams can be checked on different requirements with different ways of aggregation depending on the context. It is not practical to aggregate the data beforehand. Therefore, aggregation is a key feature for the specification language.

Entity: An entity requirement specifies the variable of interest, such as, “noise level”, “energy consumption”, etc.

Comparison: Comparison requirements usually specify the threshold of the variable, e.g., “up to 85°F”. In other cases, it also defines true or false, e.g., “the street is blocked”.

Condition: In a broad definition, conditions specify the condition or special cases of the requirement, such as, “if/unless”, “until”, and “except”.

6.3 Formalizing Temporal-Spatial Requirements

SaSTL extends STL with two spatial operators: a *spatial aggregation* operator and a *neighborhood counting* operator. Spatial aggregation enables combining (according to a chosen operation) measurements of the same type (e.g., environmental temperature), but taken from different locations. The use of this operator can be suitable in requirements where it is necessary to evaluate the average, best or worst value of a signal measurement in an area close to the desired location. The neighborhood counting operator allows measuring the number/percentage of neighbors of a location that satisfy a certain requirement.

6.3.1 SaSTL Syntax

We define a multi-dimensional *spatial-temporal signal* as $\omega : \mathbb{T} \times L \to \{\mathbb{R} \cup \{\perp\}\}^n$, where $\mathbb{T} = \mathbb{R}_{\geq 0}$, represents the continuous time and L is the set of locations. We define $X = \{x_1, \cdots, x_n\}$ as the set of variables for each location. Each variable can assume a real value $v \in \mathbb{R}$ or is undefined for a particular location ($x_i = \perp$). We denote by $\pi_{x_i}(\omega)$ as the projection of ω on its component variable $x_i \in X$. We define $P = \{p_1, \cdots, p_m\}$ a set of propositions (e.g. $\{\mathsf{School}, \mathsf{Street}, \mathsf{Hospital}, \cdots\}$) and $\mathcal{L}$ a labeling function $\mathcal{L} : L \to 2^P$ that assigns for each location the set of the propositions that are true in that location.

A *weighted undirected graph* is a tuple $G = (L, E, \eta)$ where L is a finite non-empty set of nodes representing locations, $E \subseteq L \times L$ is the set of edges connecting nodes, and $\eta : E \to \mathbb{R}_{\geq 0}$ is a cost function over edges. We define the *weighted distance* between two locations $l, l' \in L$ as

$$d(l, l') \coloneqq \min\{\sum_{e \in \sigma} \eta(e) \mid \sigma \text{ is a path between } l \text{ and } l'\}.$$

Then we define the spatial domain $\mathcal{D}$ as,

$$\begin{aligned} \mathcal{D} \coloneqq &\quad ([d_1, d_2], \psi) \\ \psi \coloneqq &\quad \top \mid p \mid \neg\, \psi \mid \psi \ \vee \ \psi \end{aligned}$$

where $[d_1, d_2]$ defines a spatial interval with $d_1 < d_2$ and $d_1, d_2 \in \mathbb{R}$, and ψ specifies the property over the set of propositions that must hold in each location. Intuitively, it draws two circles with radius $r_1 = d_1$ and $r_2 = d_2$, and the locations $l \vDash \psi$ between these two circles are selected. In particular, $\mathcal{D} = ([0, +\infty), \top)$ indicates the whole spatial domain. We denote $L^l_{([d_1,d_2],\psi)} \coloneqq \{l' \in L \mid 0 \leq d_1 \leq d(l, l') \leq d_2$ and $\mathcal{L}(l') \vDash \psi\}$ as the set of locations at a distance between d_1 and d_2 from l for which $\mathcal{L}(l')$ satisfies ψ. We denote the set of non-null values for signal variable x at time point t location l over locations in $L^l_{\mathcal{D}}$ by

$$\alpha^x_{\mathcal{D}}(\omega, t, l) \coloneqq \{\pi_x(\omega)[t, l'] \mid l' \in L^l_{\mathcal{D}} \text{ and } \pi_x(\omega)[t, l'] \neq \perp\}.$$

We define a set of operations $\mathrm{op}(\alpha^x_{\mathcal{D}}(\omega, t, l))$ for $\mathrm{op} \in \{\max, \min, \mathrm{sum}, \mathrm{avg}\}$ when $\alpha^x_{\mathcal{D}}(\omega, t, l) \neq \varnothing$ that computes the maximum, minimum, summation and average of values in the set $\alpha^x_{\mathcal{D}}(\omega, t, l)$, respectively. To be noted, Graph G and its weights between nodes are constructed flexibly based on

the property of the system. For example, we can build a graph with fully connected sensor nodes and their Euclidean distance as the weights when monitoring the air quality in a city; or we can also build a graph that only connects the street nodes when the two streets are contiguous and apply Manhattan distance. It does not affect the syntax and semantics of SaSTL.

The syntax of SaSTL is given by

$$\varphi := \quad x \sim c \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \mathcal{U}_I \varphi_2 \mid \mathcal{A}^{\mathrm{op}}_{\mathcal{D}} x \sim c \mid \mathcal{C}^{\mathrm{op}}_{\mathcal{D}} \varphi \sim c$$

where $x \in X$, $\sim \in \{<, \leq\}$, $c \in \mathbb{R}$ is a constant, $I \subseteq \mathbb{R}_{>0}$ is a real positive dense time interval, $\mathcal{U}_I$ is the *bounded until* temporal operators from STL. The *always* (denoted $\Box$) and *eventually* (denoted $\Diamond$) temporal operators can be derived the same way as in STL, where $\Diamond\varphi \equiv \mathsf{true}\ \mathcal{U}_I \varphi$, and $\Box\varphi \equiv \neg\Diamond\neg\varphi$.

In SaSTL, we define a set of spatial *aggregation* operators $\mathcal{A}^{\mathrm{op}}_{\mathcal{D}} x \sim c$ for $\mathrm{op} \in \{\max, \min, \mathrm{sum}, \mathrm{avg}\}$ that evaluate the aggregated product of traces $\mathrm{op}(\alpha^x_{\mathcal{D}}(\omega, t, l))$ over a set of locations $l \in L^l_{\mathcal{D}}$. We also define a set of new spatial *counting* operators $\mathcal{C}^{\mathrm{op}}_{\mathcal{D}} \varphi \sim c$ for $\mathrm{op} \in \{\max, \min, \mathrm{sum}, \mathrm{avg}\}$ that counts the satisfaction of traces over a set of locations. More precisely, we define $\mathcal{C}^{\mathrm{op}}_{\mathcal{D}} \varphi = \mathrm{op}(\{g((\omega, t, l') \vDash \varphi) \mid l' \in L^l_{\mathcal{D}}\})$, where $g((\omega, t, l) \vDash \varphi)) = 1$ if $(\omega, t, l) \vDash \varphi$, otherwise $g((\omega, t, l) \vDash \varphi)) = 0$. From the new *counting* operators, we also derive the *everywhere* operator as $\boxdot_{\mathcal{D}} \varphi \equiv \mathcal{C}^{\min}_{\mathcal{D}} \varphi > 0$, and *somewhere* operator as $\diamondsuit_{\mathcal{D}} \varphi \equiv \mathcal{C}^{\max}_{\mathcal{D}} \varphi > 0$. In addition, $\mathcal{C}^{\mathrm{sum}}_{\mathcal{D}} \varphi$ specifies the total number of locations that satisfy φ and $\mathcal{C}^{\mathrm{avg}}_{\mathcal{D}} \varphi$ specifies the percentage of locations satisfying φ.

We now illustrate how to use SaSTL to specify various city requirements, especially for the spatial aggregation and spatial counting, and how important these operators are for the smart city requirements using examples below.

Example 6.1 (Spatial Aggregation)**.** *Assume we have a requirement, "The average noise level in the school area (within 1 km) in New York City should always be less than 50 dB and the worst should be less than 80 dB in the next 3 hours" is formalized as,* $\boxdot_{([0,+\infty),\mathsf{School})} \Box_{[0,3]} ((\mathcal{A}^{\mathrm{avg}}_{([0,1],\top)} x_{\mathsf{Noise}} < 50) \wedge (\mathcal{A}^{\mathsf{max}}_{([0,1],\top)} x_{\mathsf{Noise}} < 80))$*.* $([0, +\infty), \mathsf{School})$ *selects all the locations labeled as "school" within the whole New York city (*$[0, +\infty)$*) (predefined by users).* $\Box_{[0,3]}$ *indicates this requirement is valid for the next three hours.* $(\mathcal{A}^{\mathrm{avg}}_{([0,1],\top)} x_{\mathsf{Noise}} < 50) \wedge (\mathcal{A}^{\mathsf{max}}_{([0,1],\top)} x_{\mathsf{Noise}} < 80)$ *calculates the average and maximal values in 1 km for each "school", and compares them with the requirements, i.e. 50 dB and 80 dB.*

Without the spatial aggregation operators, STL and its extended languages cannot specify this requirement. First, they are not able to first dynamically find all the locations labeled as "school". To monitor the same spatial range, users have manually get all traces from schools, and then repeatedly apply this requirement to each located sensor within 1 km of a school and do the same for all schools. More importantly, STL and its extended languages could not specify "average" or "worst" noise level. Instead, it only monitors each single value, which is prone to noises and outliers and thereby causes inaccurate results.

Example 6.2 (Spatial Counting)**.** *A requirement that "At least 90% of the streets, the particulate matter (PMx) emission should not exceed Moderate in 2 hours" is formalized as* $\mathcal{C}^{\text{avg}}_{([0,+\infty),\mathsf{Street})}(\square_{[0,2]}(x_{\mathsf{PMx}} < \mathsf{Moderate})) > 0.9$*.* $\mathcal{C}^{\text{avg}}_{([0,+\infty),\mathsf{Street})}\varphi > 0.9$ *represents the percentage of satisfaction is larger than 90%. Specifying the percentage of satisfaction is very common and important among city requirements.*

6.3.2 SaSTL Semantics

We define the SaSTL semantics as the *satisfiability relation* $(\omega, t, l) \vDash \varphi$, indicating that the spatio-temporal signal ω satisfies a formula φ at the time point t in location l when $\pi_v(\omega)[t,l] \neq \perp$ and $\alpha^x_{\mathcal{D}}(\omega,t,l) \neq \varnothing$. We define that $(\omega,t,l) \vDash \varphi$ if $\pi_v(\omega)[t,l] = \perp$.

$$
\begin{aligned}
(\omega,t,l) &\vDash x \sim c && \Leftrightarrow \pi_x(\omega)[t,l] \sim c \\
(\omega,t,l) &\vDash \neg\varphi && \Leftrightarrow (\omega,t,l) \nvDash \varphi \\
(\omega,t,l) &\vDash \varphi_1 \wedge \varphi_2 && \Leftrightarrow (\omega,t,l) \vDash \varphi_1 \text{ and } (\omega,t,l) \vDash \varphi_2 \\
(\omega,t,l) &\vDash \varphi_1 \mathcal{U}_I \varphi_2 && \Leftrightarrow \exists t' \in (t+I) \cap \mathbb{T} : (\omega,t',l) \vDash \varphi_2 \text{ and } \forall t'' \in (t,t'), (\omega,t'',l) \vDash \varphi_1 \\
(\omega,t,l) &\vDash \mathcal{A}^{\text{op}}_{\mathcal{D}} x \sim c && \Leftrightarrow \text{op}(\alpha^x_{\mathcal{D}}(\omega,t,l)) \sim c \\
(\omega,t,l) &\vDash \mathcal{C}^{\text{op}}_{\mathcal{D}} \varphi \sim c && \Leftrightarrow \text{op}(\{g((\omega,t,l') \vDash \varphi) \mid l' \in L^l_{\mathcal{D}}\}) \sim c
\end{aligned}
$$

where, for counting operator $(\omega,t,l) \vDash \mathcal{C}^{\text{op}}_{\mathcal{D}}\varphi \sim c$, the valid ranges for c are $c \in [0,1)$ when $\mathsf{op} = \mathsf{sum/min}$, and $c \in [0,N]$ when $\mathsf{op} = \mathsf{sum/min}$. Otherwise (e.g., $c < 0$), the requirement is trivially satisfied or violated.

Example 6.3. *Following Example 6.1, checking the city states with a requirement,*

$\boxdot_{([0,+\infty),\mathsf{Hospital})}\ \square_{[0,5]}\ ((\mathcal{A}^{\mathrm{avg}}_{([0,500],\top)}x_{\mathsf{AQI}} < 50) \wedge (\mathcal{A}^{\mathsf{max}}_{([0,500],\top)}x_{\mathsf{AQI}} < 80))$,

to start with, assuming we have the AQI level data from a number of sensors within 500 meters of one of the hospital, the sensor readings in 5 hours as, {[51, ..., 11], [80, ..., 30],..., [40, ..., 30]}, $\varphi_t = (\mathcal{A}^{\mathrm{avg}}_{([0,500],\top)]}x_{\mathsf{AQI}} < 50) \wedge (\mathcal{A}^{\mathsf{max}}_{([0,500],\top)}x_{\mathsf{AQI}} < 80)$*, then, we check* φ_t *for this hospital at each time,*

- *at* $t = 1$, $\mathrm{avg}(51, ..., 40) > 50 \wedge \mathsf{max}(51, ..., 40) < 80$, *thus,* $\varphi_{t1} = False$,
- *at* $t = 2$, $\mathrm{avg}(49, ..., 20) < 50 \wedge \mathsf{max}(49, ..., 20) > 80$, *thus,* $\varphi_{t1} = False$,
- ...
- *at* $t = 5, \mathrm{avg}(11, ..., 30) < 50 \wedge \mathsf{max}(11, ..., 30) < 80$, *thus,* $\varphi_{t1} = True$.

Thus, we have $\square_{[0,5]}\varphi_t = False$.

Next, the monitor checks all qualified hospitals the same way and reaches the final results,

$$\boxdot_{([0,+\infty),\mathsf{Hospital})}\ \square_{[0,5]}\ ((\mathcal{A}^{\mathrm{avg}}_{([0,500],\top)}x_{\mathsf{AQI}} < 50) \wedge (\mathcal{A}^{\mathsf{max}}_{([0,500],\top)}x_{\mathsf{AQI}} < 80)) = False.$$

In a real scenario, the monitor algorithm can also decide to terminate the monitor and return the False result when at $t = 1$, because the always operator returns False as long as a one-time violation occurs. Similarly, the everywhere operator will also return False when the first hospital violates the requirement.

Definition 6.1 (Quantitative Semantics)**.** *Let* $x > c$ *be a numerical predicate, we then define the robustness degree (i.e. the quantitative satisfaction) function* $\rho(\varphi, \omega, t, l)$ *for an SaSTL formula over a spatial-temporal signal* ω *as,*

$$
\begin{aligned}
\rho(x \sim c, \omega, t, l) &= \pi_x(\omega)[t, l] - c \\
\rho(\neg\varphi, \omega, t, l) &= -\rho(\varphi, \omega, t, l) \\
\rho(\varphi_1 \vee \varphi_2, \omega, t, l) &= \max\{\rho(\varphi_1, \omega, t, l), \rho(\varphi_2, \omega, t, l)\} \\
\rho(\varphi_1 \mathcal{U}_I \varphi_2, \omega, t, l) &= \sup_{t' \in (t+I) \cap \mathbb{T}}(\min\{\rho(\varphi_2, \omega, t', l), \inf_{t'' \in [t, t']}(\rho(\varphi_1, \omega, t'', l))\})
\end{aligned}
$$

$$
\rho(\mathcal{A}^{\text{op}}_{\mathcal{D}} x \sim c, \omega, t, l) = \begin{cases} \frac{\mathsf{sum}(\alpha^x_{\mathcal{D}}(\omega,t,l)) - c}{|\alpha^x_{\mathcal{D}}(\omega,t,l)|} & \mathsf{op} = \mathsf{sumop}(\alpha^x_{\mathcal{D}}(\omega, t, l)) - c \quad \mathsf{op} \in \{\mathsf{max}, \mathsf{min}, \mathsf{avg}\} \end{cases}
$$

$$
\rho(\mathcal{C}^{\text{op}}_{\mathcal{D}} \varphi \sim c, \omega, t, l) = \begin{cases} \max_{l' \in L^l_{\mathcal{D}}}\{\rho(\varphi, \omega, t, l')\} & \mathsf{op} = \mathsf{max} \\ \min_{l' \in L^l_{\mathcal{D}}}\{\rho(\varphi, \omega, t, l')\} & \mathsf{op} = \mathsf{min} \\ \delta(\lceil c \rceil, \{\rho(\varphi, \omega, t, l') \mid l' \in L^l_{\mathcal{D}}\}) & \mathsf{op} = \mathsf{sum} \\ \delta(\lceil c \times |L^l_{\mathcal{D}}| \rceil, \{\rho(\varphi, \omega, t, l') \mid l' \in L^l_{\mathcal{D}}\}) & \mathsf{op} = \mathsf{avg} \end{cases}
$$

where we define $\delta(k, S)$ as a function that returns the kth smallest number of set S, $|S| > 0$, and $0 \leq k \leq |S|$. For $\mathcal{C}^{\text{op}}_{\mathcal{D}} \varphi \sim c$, when $\mathsf{op} = \mathsf{sum}$, it requires that there are at least $\lceil c \rceil$ locations that satisfy φ, thus, we denote the $\lceil c \rceil$th smallest robustness value from $\{\rho(\varphi, \omega, t, l') \mid l' \in L^l_{\mathcal{D}}\}$ as the robustness value of this formula. $[c]$ indicates the smallest integer that is larger than or equal to c. Similarly, when $\mathsf{op} = \mathsf{avg}$, the formula is converted as there are at least $\lceil c \times |L^l_{\mathcal{D}}| \rceil$ locations that satisfy φ, thus, we denote the $\lceil c \times |L^l_{\mathcal{D}}| \rceil$th smallest robustness value from $\{\rho(\varphi, \omega, t, l') \mid l' \in L^l_{\mathcal{D}}\}$ as the robustness value of this formula. Same as the Boolean semantics, the valid ranges for c are $c \in [0, 1)$ when $\mathsf{op} = \mathsf{sum/min}$, and $c \in [0, N]$ when $\mathsf{op} = \mathsf{sum/min}$. Otherwise (e.g., $c < 0$), the requirement is trivially satisfied or violated.

Example 6.4. *Assuming we have data of traffic counts (1,2,3), (2,3,4), (4,5,7) from three locations satisfying $\mathcal{D}$, thus,*

- $\rho(\mathcal{C}^{\mathsf{max}}_{\mathcal{D}}(\Box_{[0,2]}(x > 5)) > 0) = \rho(\mathcal{C}^{\mathsf{max}}_{\mathcal{D}}(\{-4, -3, 2\}) > 0) = 2$
- $\rho(\mathcal{C}^{\mathsf{min}}_{\mathcal{D}}(\Box_{[0,2]}(x > 5)) > 0) = \rho(\mathcal{C}^{\mathsf{min}}_{\mathcal{D}}(\{-4, -3, 2\}) > 0) = -4$
- $\rho(\mathcal{C}^{\mathsf{sum}}_{\mathcal{D}}(\Box_{[0,2]}(x > 5)) > 1) = \rho(\mathcal{C}^{\mathsf{sum}}_{\mathcal{D}}(\{-4, -3, 2\}) > 1) = -3$
- $\rho(\mathcal{C}^{\mathsf{avg}}_{\mathcal{D}}(\Box_{[0,2]}(x > 5)) > 0.2) = \rho(\mathcal{C}^{\mathsf{avg}}_{\mathcal{D}}(\{-4, -3, 2\}) > 0.2) = 2$

The quantitative semantics of SaSTL inherit the two fundamental properties of STL, i.e., soundness and correctness. We give the formal definitions below.

Theorem 6.1 (Soundness)**.** *Let φ be an STL formula, ω a trace and t a time,*

$$\begin{aligned}\rho(\varphi,\omega,t,l) > 0 \quad &\Rightarrow (\omega,t,l) \vDash \varphi \\ \rho(\varphi,\omega,t,l) < 0 \quad &\Rightarrow (\omega,t,l) \nvDash \varphi\end{aligned}$$

Proof. We prove the first property $\rho(\varphi,\omega,t,l) > 0 \Rightarrow (\omega,t,l) \vDash \varphi$ by induction:

First we show the soundness property hold for the predicate $\varphi := \mu$. In this case, we have $\rho(\varphi,\omega,t,l) = f(x)$. Therefore, if $\rho(\varphi,\omega,t,l) > 0$ we have $f(x) > 0$, that is, $(\omega,t,l) \vDash \varphi$.

Case $\varphi = \neg\varphi'$: We have $\rho(\varphi,\omega,t) = -\rho(\varphi',\omega,t,l) > 0$. Therefore we have $\rho(\varphi',\omega,t,l) < 0$, that is, $(\omega,t,l) \nvDash \varphi'$, which is equivalent to $(\omega,t,l) \vDash \varphi$ by definition.

Case $\varphi = \varphi_1 \wedge \varphi_2$: We have $\rho(\varphi_1 \wedge \varphi_2,\omega,t,l) = \min\{\rho(\varphi_1,\omega,t,l),\rho(\varphi_2,\omega,t,l)\} > 0$. Therefore, we have $\rho(\varphi_1,\omega,t,l) > 0$ and $\rho(\varphi_1,\omega,t,l) > 0$. Thus, $(\omega,t,l) \vDash \varphi_1$ and $(\omega,t,l) \vDash \varphi_2$. By definition, we have $(\omega,t,l) \vDash \varphi$.

Case $\varphi = \varphi_1 \mathcal{U}_I \varphi_2$: $\rho = \max\limits_{t' \in (t,t+I)} \{\min\{\rho(\varphi_2,\omega,t',l), \min\limits_{t'' \in (t,t')} \rho(\varphi_1,\omega,t'',l)\}\} > 0$. We have $\exists t' \in (t + I), \min\{\rho(\varphi_2,\omega,t',l), \min\limits_{t'' \in (t,t')} \rho(\varphi_1,\omega,t'',l)\} > 0$. Therefore, $\exists t' \in (t+I), \rho(\varphi_2,\omega,t',l) > 0 \wedge \min\limits_{t'' \in (t,t')} \rho(\varphi_1,\omega,t'',l) > 0$. Thus, it's equivalent to $\exists t' \in (t+I) \cap \mathbb{T}, (\omega,t',l) \vDash \varphi_2$ and $\forall t'' \in (t,t'), (\omega,t'',l) \vDash \varphi_1$. By definition, we have $(\omega,t,l) \vDash \varphi$.

Case $\varphi = \mathcal{A}^{\text{op}}_{\mathcal{D}} x \sim c$: we have $\rho(\mathcal{A}^{\text{op}}_{\mathcal{D}} x \sim c,\omega,t,l) > 0$, which indicates $\text{op}(\alpha^x_{\mathcal{D}}(\omega,t,l)) - c > 0$, following the definition, we have $(\mathcal{A}^{\text{op}}_{\mathcal{D}} x \sim c,\omega,t,l) \vDash \varphi$.

Case $\varphi = \mathcal{C}^{\text{op}}_{\mathcal{D}} \varphi \sim c$ when $\mathsf{op} = \mathsf{max}$, we have $\max_{l' \in L^l_{\mathcal{D}}} \{\rho(\varphi,\omega,t,l')\} > 0$, thus, there is at least one location $l \in \mathcal{D}$,$\rho(\varphi,\omega,t,l) > 0$, i.e., $(\omega,t,l) \vDash \varphi$, therefore, $\mathsf{max}(\{g((\omega,t,l') \vDash \varphi) \mid l' \in L^l_{\mathcal{D}}\}) > c$ $(c \in [0,1))$ is true, therefore, $(\omega,t,l) \vDash \mathcal{C}^{\mathsf{max}}_{\mathcal{D}} \varphi > c$. when $\mathsf{op} = \mathsf{min}$, we have $\min_{l' \in L^l_{\mathcal{D}}} \{\rho(\varphi,\omega,t,l')\} > 0$, thus, for any location, $\rho(\varphi,\omega,t,l) > 0$, i.e., $l \in \mathcal{D}$, $(\omega,t,l) \vDash \varphi$, therefore, $\mathsf{min}(\{g((\omega,t,l') \vDash \varphi) \mid l' \in L^l_{\mathcal{D}}\}) > c$ $(c \in [0,1))$ is true, therefore, $(\omega,t,l) \vDash \mathcal{C}^{\mathsf{min}}_{\mathcal{D}} \varphi \sim c$. When $\mathsf{op} = \mathsf{sum}$, we have $\delta(\lceil c \rceil, \{\rho(\varphi,\omega,t,l') \mid l' \in L^l_{\mathcal{D}}\}) > 0$, thus, for at least $\lceil c \rceil$ locations l, we have $\rho(\varphi,\omega,t,l) \mid l \in L^l_{\mathcal{D}} > 0$, i.e., $\mathsf{sum}(\{g((\omega,t,l) \vDash \varphi) \mid l \in L^l_{\mathcal{D}}\}) > c$ is true, therefore, $(\omega,t,l) \vDash \mathcal{C}^{\mathsf{sum}}_{\mathcal{D}} \varphi > c$. Similarly, we can prove when $\mathsf{op} = \mathsf{avg}$, if $\delta(\lceil c \times |L^l_{\mathcal{D}}| \rceil, \{\rho(\varphi,\omega,t,l') \mid l' \in L^l_{\mathcal{D}}\}) > 0$, then $(\omega,t,l) \vDash \mathcal{C}^{\mathsf{avg}}_{\mathcal{D}} \varphi > c$.

□

Secondly, if ω satisfies φ at time t, any other trace ω' whose point-wise distance from ω is smaller than $\rho(\varphi, \omega, t, l)$ also satisfies φ at time t.

Theorem 6.2 (Correctness)**.** *Let φ be an STL formula, ω and ω' traces over the same time and spatial domains, and $t, l \in dom(\varphi, \omega)$, then*

$$(\omega, t, l) \vDash \varphi \text{ and } \|\omega - \omega'\|_\infty < \rho(\varphi, \omega, t, l) \Rightarrow (\omega', t, l) \vDash \varphi$$

Proof. By induction, we have the following cases:

Case $\varphi := x \sim c$: We have $\rho(\varphi, \omega', t, l) = \pi_x(\omega')[t, l] - c \geq \pi_x(\omega)[t, l] - c - \|\omega - \omega'\|_\infty = \rho(\varphi, \omega, t, l) - \|\omega - \omega'\|_\infty > 0$. Therefore, we have $(\omega', t, l) \vDash \varphi$.

Case $\varphi := \neg\varphi'$: We have $\rho(\varphi, \omega', t, l) = -\rho(\varphi', \omega', t, l)$. By the inductive assumption we have $\rho(\varphi', \omega', t, l) < 0$. Therefore, we have $(\omega', t, l) \vDash \varphi$.

Case $\varphi := \varphi_1 \vee \varphi_2$: Following the condition, we have either $(\omega, t, l) \vDash \varphi_1$ holds or $(\omega, t, l) \vDash \varphi_2$ holds. We also have $\rho(\varphi, \omega', t, l) = \max\{\rho(\varphi_1, \omega', t, l), \rho(\varphi_2, \omega', t, l)\}$. If $(\omega, t, l) \vDash \varphi_1$, by the inductive assumption we have $\rho(\varphi_1, \omega', t, l) > 0$. Therefore, $\rho(\varphi, \omega, t, l) > 0$. Similarly, if $(\omega, t, l) \vDash \varphi_2$, by the inductive assumption we have $\rho(\varphi_2, \omega', t, l) > 0$. Therefore, we have $(\omega', t, l) \vDash \varphi$.

Case $\varphi = \varphi_1 \mathcal{U}_I \varphi_2$: As $(\omega, t, l) \vDash \varphi$, there exists t' that $\forall t'' \in (t, t'), \rho(\varphi_1, \omega, t'', l) \geq \rho(\varphi, \omega, t, l)$ and $\rho(\varphi_2, \omega, t', l) \geq \rho(\varphi, \omega, t, l)$. By the inductive assumption, we have $(\omega', t', l) \vDash \varphi_2$ and $\forall t'' \in (t, t'), (\omega', t'', l) \vDash \varphi_1$. Therefore, we have $(\omega', t, l) \vDash \phi$.

Case $\varphi = \mathcal{A}^{\text{op}}_{\mathcal{D}} x \sim c$:

- When $\mathsf{op} = \mathsf{sum}$, $\rho(\phi, \omega', t, l) = \frac{\mathsf{sum}(\alpha^x_{\mathcal{D}}(\omega', t, l)) - c}{|\alpha^x_{\mathcal{D}}(\omega', t, l)|} \geq \frac{\mathsf{sum}(\alpha^x_{\mathcal{D}}(\omega, t, l)) - c - \sum_{d \in \alpha^x_{\mathcal{D}}(\omega, t, l)} \|\omega - \omega'\|_\infty}{|\alpha^x_{\mathcal{D}}(\omega, t, l)|} = \rho(\phi, \omega, t, l) - \|\omega - \omega'\|_\infty > 0$. Therefore, we have $(\omega', t, l) \vDash \phi$.

- When $\mathsf{op} \neq \mathsf{sum}$, we first show that $\mathsf{op}(\alpha^x_{\mathcal{D}}(\omega, t, l)) - \mathsf{op}(\alpha^x_{\mathcal{D}}(\omega', t, l)) \leq \|\omega - \omega'\|_\infty$. Recall the definition that $\alpha^x_{\mathcal{D}}(\omega, t, l) := \{\pi_x(\omega)[t, l'] \mid l' \in L^l_{\mathcal{D}} \text{ and } \pi_x(\omega)[t, l'] \neq \perp\}$. For any combination of t and l, $\pi_x(\omega)[t, l] \leq \pi_x(\omega')[t, l] + \|\omega - \omega'\|_\infty$. As all the items of $\alpha^x_{\mathcal{D}}(\omega, t, l)$ holds the property, for the operators max, min and avg, $\mathsf{op}(\alpha^x_{\mathcal{D}}(\omega, t, l)) - \mathsf{op}(\alpha^x_{\mathcal{D}}(\omega', t, l)) \leq \|\omega - \omega'\|_\infty$.

Therefore we have $\rho(\phi,\omega',t,l) = \mathsf{op}(\alpha^x_{\mathcal{D}}(\omega',t,l)) - c \geq \mathsf{op}(\alpha^x_{\mathcal{D}}(\omega,t,l)) - \|\omega-\omega'\|_\infty - c = \rho(\phi,\omega,t,l) - \|\omega-\omega'\|_\infty > 0$, which indicates $(\omega',t,l) \vDash \phi$.

Case $\varphi = \mathcal{C}^{\mathrm{op}}_{\mathcal{D}}\varphi' \sim c$:

- When $\mathsf{op} = \mathsf{sum}$, as $\rho(\mathcal{C}^{\mathrm{op}}_{\mathcal{D}}\varphi' \sim c,\omega,t,l) = \delta(\lceil c \rceil, \{\rho(\varphi',\omega',t,l') \mid l' \in L^l_{\mathcal{D}}\})$, we know that there exists at least $\lceil c \rceil$ different $l' \in L^l_{\mathcal{D}}$ that $\rho(\varphi',\omega,t,l') \geq \rho(\mathcal{C}^{\mathrm{op}}_{\mathcal{D}}\varphi' \sim c,\omega,t,l) > \|\omega-\omega'\|_\infty$. By the inductive rule, we have at least $\lceil c \rceil$ different $l' \in L^l_{\mathcal{D}}$ that $\rho(\varphi',\omega',t,l') > 0$. Therefore, by the defintion of $\rho(\mathcal{C}^{\sum}_{\mathcal{D}}\varphi' \sim c)$ of we have $(\omega',t,l) \vDash \phi$.

- Similarly when $\mathsf{op} = \mathsf{avg}$, as $\rho(\mathcal{C}^{\mathrm{op}}_{\mathcal{D}}\varphi' \sim c,\omega,t,l) = \delta(\lceil c \times |L^l_{\mathcal{D}}| \rceil \{\rho(\varphi',\omega',t,l') \mid l' \in L^l_{\mathcal{D}}\})$, we know that there exists at least $\lceil c \times |L^l_{\mathcal{D}}| \rceil$ different $l' \in L^l_{\mathcal{D}}$ that $\rho(\varphi',\omega,t,l') \geq \rho(\mathcal{C}^{\mathrm{op}}_{\mathcal{D}}\varphi' \sim c,\omega,t,l) > \|\omega-\omega'\|_\infty$. By the inductive rule, we have at least $\lceil c \times |L^l_{\mathcal{D}}| \rceil$ different $l' \in L^l_{\mathcal{D}}$ that $\rho(\varphi',\omega',t,l') > 0$. Therefore, we have $(\omega',t,l) \vDash \phi$.

- When $\mathsf{op} = \mathsf{max}$, $\rho(\mathcal{C}^{\mathrm{op}}_{\mathcal{D}}\varphi' \sim c,\omega,t,l) = \max_{l' \in L^l_{\mathcal{D}}}\{\rho(\varphi,\omega,t,l')\}$. Let l' be the location that $\rho(\varphi,\omega,t,l')$ achieves maximum, we have $\rho(\varphi',\omega,t,l') \geq \rho(\mathcal{C}^{\mathrm{op}}_{\mathcal{D}}\varphi' \sim c,\omega,t,l) > \|\omega-\omega'\|_\infty$. By the inductive rule, $\rho(\varphi',\omega',t,l') > 0$. Therefore, we have $(\omega',t,l) \vDash \phi$.

- When $\mathsf{op} = \mathsf{min}$, $\rho(\mathcal{C}^{\mathrm{op}}_{\mathcal{D}}\varphi' \sim c,\omega,t,l) = \min_{l' \in L^l_{\mathcal{D}}}\{\rho(\varphi,\omega,t,l')\}$. We have for every $l' \in L^l_{\mathcal{D}}$, $\rho(\varphi',\omega,t,l') \geq \rho(\mathcal{C}^{\mathrm{op}}_{\mathcal{D}}\varphi' \sim c,\omega,t,l) > \|\omega-\omega'\|_\infty$. By the inductive rule, We have for every $l' \in L^l_{\mathcal{D}}$ that $\rho(\varphi',\omega',t,l') > 0$. Therefore, we have $(\omega',t,l) \vDash \phi$.

□

In summary, the qualitative value indicates if the signal (i.e. city data) satisfies the requirement. The quantitative value indicates the satisfaction or dissatisfaction degree. If it is larger or equal than zero, it means that the requirement is satisfied. The larger the value is, the more the requirement is satisfied. On the contrary, if the value is smaller than zero, it means the requirement is not satisfied. The smaller the value is, the more the requirement is dissatisfied.

6.4 Efficient Monitoring for SaSTL

In this section, we first present both Boolean and quantitative monitoring algorithms for SaSTL, then describe two optimization methods to speed up the monitoring performance.

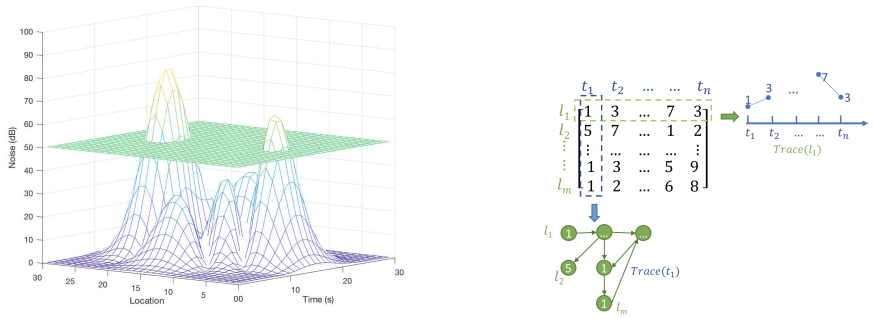


Figure 6.2: 2-dimension city data over spatial and temporal domains. ((1) shows the noise level over a period of time and across multiple locations, x: location, y: time (s), z: noise (dB), (2) shows the matrix of the data, each row is a time-series data from one location and each column is a set of data from all locations at one time-stamp.)

6.4.1 Monitoring Algorithms for SaSTL

The *inputs* of the monitor are the SaSTL requirements φ (including the time t and location l), a weighted undirected graph G and the temporal-spatial data ω. In smart cities, the data on city states is collected continuously or periodically. It is a 2-dimension data, as illustrated in Figure 6.2. Figure 6.2 (1) shows the noise level over a period of time and across multiple locations, x, y, and z axes indicate the location, time, and noise, respectively. Figure 6.2 (2) shows the matrix of the data, each row is a time-series data from one location and each column is a set of data from all locations at one time-stamp. In Figure 6.2 (1), the points above the plane violate the requirements and the amount over the plane represents the level of violation.

For the Boolean monitoring algorithm, the *output* for each requirement is a Boolean value indicating whether the requirement is satisfied or not. For the quantitative monitoring algorithm (Algorithm 6.4), the *output* for each requirement is a number indicating the satisfaction degree of the requirement. To start with, the monitoring algorithm parses φ to sub-formulas and calculates the satisfaction for each operation recursively. We derived operators $\Box$ and $\Diamond$ from $\mathcal{U}_I$, and operators $\boxdot$ and $\diamondsuit$ from $\mathcal{C}^{\mathsf{op}}_{\mathcal{D}} \sim c$, so we only show the algorithms for $\mathcal{U}_I$ and $\mathcal{C}^{\mathsf{op}}_{\mathcal{D}} \sim c$.

We present the quantitative monitoring algorithms of the operators $\mathcal{A}^{\mathsf{op}}_{\mathcal{D}}$ and $\mathcal{C}^{\mathsf{op}}_{\mathcal{D}}$ in Algorithm 6.5 and Algorithm 6.6, respectively. We apply distributed parallel algorithm $\mathsf{deScan}()$ [127] to accelerate the process of searching locations that satisfy $\mathcal{D}$. As we can tell from the algorithms, essentially, $\mathcal{A}^{\mathsf{op}}_{\mathcal{D}}$ calculates the aggregated values on the signal over a spatial domain, while $\mathcal{C}^{\mathsf{op}}_{\mathcal{D}}$ calculates the

Algorithm 6.1: SaSTL Boolean monitoring algorithm MonitorB$(\varphi, \omega, t, l, G)$

```
Input  : SaSTL Requirement φ, Signal ω, Time t, Location l, weighted undirected graph G
Output : Boolean Satisfaction Value
begin
    switch φ do
        Case x ∼ c
            return π_x(ω)[t, l] ∼ c;
        Case ¬φ
            return ¬ MonitorB (φ, ω, t, l, G);
        Case φ1 ∧ φ2                                   ▷ *[r]See Alg. 6.7 for an update
            return MonitorB (φ1, ω, t, l, G) ∧ MonitorB (φ2, ω, t, l, G)
        Case φ1 U_I φ2
            Boolean f := True;
            for t' ∈ (t + I) ∩ T do
                if Monitor(φ2, ω, t', l, G) then
                    f := True;
                    for t'' ∈ [t, t'] do
                        f := f ∧ Monitor(φ1, ω, t'', l, G);
                        if (¬f) then break;
                    end
                    if (f) then return True;
                end
            end
            return False;
        Case A^op_D x ∼ c                              ▷ *[r]See Alg. 6.2
            return AggregateB(x, c, op, D, t, l, G);
        Case C^op_D φ ∼ c                              ▷ *[r]See Alg. 6.3 and Alg. 6.8
            return CountingNeighboursB(φ, c, op, D, t, l, G);
    end
end
```

aggregated results over spatial domain. For the quantitative monitoring algorithm (as presented in Algorithm 6.4), the *output* for each requirement is a robustness value indicating its satisfaction degree. Similar to the Boolean monitoring algorithm, the quantitative monitoring algorithm also parses φ to sub-formulas and calculates the satisfaction for each operation recursively. We present the outline of quantitative monitoring algorithm in Algorithm 6.4.

The time complexity of monitoring the logical and temporal operators of SaSTL is the same as STL [128]. The time complexity to monitor classical logical operators or basic propositions such as $\neg x$, $\wedge$ and $x \sim c$ is $O(1)$. The time complexity to monitor temporal operators such as $\Box_I$, $\Diamond_I$, $\mathcal{U}_I$ is $O(T)$, where T is the total number of samples within time interval I. In this chapter, we present the time complexity analysis for the spatial operators (Lemma 6.1) and the new SaSTL monitoring algorithm (Theorem 6.3). The total number of locations is denoted by n. We assume that the positions of the locations cannot change in time (a fixed grid). We can pre-compute all the distances between locations and store them in an array of range trees [129] (one range tree for each location). We further denote the monitored formula as ϕ, which can be represented by a syntax tree, and let $|\phi|$ denote the total number of nodes in the syntax tree (number of operators).

Lemma 6.1 (Complexity of spatial operators)**.** *The time complexity to monitor at each location l at*

Algorithm 6.2: AggregateB$(x, op, \mathcal{D}, \omega, t, l, G)$

```
Function AggregateB(x, c, op, D, ω, t, l, G):
    begin
        Real v := 0; n := 0;
        if op == "min" then v := ∞ ;
        if op == "max" then v := −∞ ;
        L^l_D := deScan(l, G, D)
        for l' ∈ L^l_D do
            if op ∈ {min, max, sum} then
                v := op(v, π_x(ω)[t, l']);
            end
            if op == "avg" then
                v := sum(v, π_x(ω)[t, l']);
            end
            n := n + 1
        end
        if op == "avg" ∧ n ≠ 0 then v := v/n ;
        if n == 0 then
            return True
        else
            return v ∼ c;
        end
    end
```

time t the satisfaction of a spatial operator such as $\boxdot_{\mathcal{D}}$, $\diamondsuit_{\mathcal{D}}$, $\mathcal{A}^{\mathsf{op}}_{\mathcal{D}}$, and $\mathcal{C}^{\mathsf{op}}_{\mathcal{D}}$ is $O(log(n) + |L|)$ where L is the set of locations at distance within the range $\mathcal{D}$ from l.

Proof. According to [129], the time complexity to retrieve a set of nodes L with a distance to a desired location in a range $\mathcal{D}$ from a location l is $O(log(n) + |L|)$. The aggregation and counting operations of Algorithm 6.2 and Algorithm 6.3 can be performed while the locations are retrieved. □

Theorem 6.3. *The time complexity of the SaSTL monitoring algorithm is upper-bounded by $O(|\phi| \times T_{max} \times (log(n) + |L|_{max}))$ where T_{max} is the largest number of samples of the intervals considered in the temporal operators of ϕ and $|L|_{max}$ is the maximum number of locations defined by the spatial temporal operators of ϕ.*

Proof. Following Lemma 6.1, by considering T_{max} the worst possible number of samples that we need to consider for all possible intervals of temporal operators present in the formula, and $|L|_{max}$ for the worst possible number of locations that we need to consider for all possible intervals of spatial operators present in the formula. When there are two or more operators nested, the time complexity for one operation is bounded by $O(T_{max}\ (log(n) + |L|_{max}))$. As there are $|\phi|$ nodes in the syntax tree of ϕ, the time complexity of the SaSTL monitoring algorithm is bounded by the summation over all $|\phi|$ nodes, which is $O(|\phi|\ T_{max}\ (log(n) + |L|_{max}))$. □

Algorithm 6.3: CountingNeighboursB$(x, op, \mathcal{D}, \omega, t, l, G)$

```
begin
    Real v := 0; n := 0
    if op == "min" then v := ∞ ;
    if op == "max" then v := −∞ ;
    L_D^l := deScan(l, G, D)
    for l' ∈ L_D^l do
        if Monitor(φ, ω, t, l, G) ∧ op ∈ {min, max, sum} then
            v := op(v, 1);
        end
        if Monitor(φ, ω, t, l, G) ∧ op == "avg" then
            v := sum(v, 1);
        end
        n := n + 1
    end
    if op == "avg" ∧n ≠ 0 then v := v/n ;
    if n == 0 then
        return True
    else
        return v ~ c;
    end
end
```

6.4.2 Optimizing SaSTL Parsing

To monitor a requirement, the first step is parsing the requirement to a set of sub formulas with their corresponding spatial-temporal ranges. Then, we calculate the results for the sub-formulas. The traditional parsing process of STL builds and calculates the syntax tree on the sequential order of the formula. It does not consider the complexity of each sub-formula. However, in many cases, especially with the PoIs specified in smart cities, checking the simpler propositional variable to quantify the spatial domain first can significantly reduce the number of temporal signals to check in a complicated formula. For example, the city abstracted graph in Figure 6.3, the large nodes represent the locations of PoIs, among which the red ones represent the schools, and blue ones represent other PoIs. The small black nodes represent the locations of data sources (e.g. sensors). Assuming a requirement $\boxdot_{([0,+\infty),\mathsf{School})} \square_{[a,b]} (\mathcal{A}^{\mathsf{op}}_{([0,d],\top])}\varphi \sim c)$ requires to aggregate and check φ only nearby schools (i.e., the red circles), but it will actually check data sources of all nearby 12 nodes if one is following the traditional parsing algorithm. In New York City, there are about 2000 primary schools, but hundreds of thousands of PoIs in total. A very large amount of computing time would be wasted in this way.

To deal with this problem, we now introduce a monitoring cost function $\mathsf{cost} : \Phi \times L \times G_L \to \mathbb{R}^+$, where Φ is the set of all the possible SaSTL formulas, L is the set of locations, G_L is the set of all the possible undirected graphs with L locations. The cost function for φ is defined as:

Algorithm 6.4: SaSTL quantitative monitoring algorithm $\mathsf{MonitorQ}(\varphi, \omega, t, l, G)$

Input : SaSTL Requirement φ, Signal ω, Time t, Location l, weighted undirected graph G
Output : Satisfaction Value ρ
begin
  switch φ **do**
    Case $x \sim c$
      return $\pi_x(\omega)[t, l] - c$;
    Case $\neg\varphi$
      return- $\mathtt{MonitorQ}(\varphi, \omega, t, l, G)$;
    Case $\varphi_1 \wedge \varphi_2$
      return $\min(\mathtt{MonitorQ}(\varphi_1, \omega, t, l, G)$,
      $\mathtt{MonitorQ}(\varphi_2, \omega, t, l, G))$;
    Case $\varphi_1 U_I \varphi_2$
      Real $v := -\infty$
      for $t' \in (t + I) \cap \mathbb{T}$ **do**
        $v' := \mathtt{MonitorQ}(\varphi_2, \omega, t', l, G)$
        for $t'' \in [t, t']$ **do**
          $v' := \min\{v', \mathtt{MonitorQ}(\varphi_2, \omega, t'', l, G)\}$
        end
        $v = \max\{v, v'\}$
      end
      return v;
    Case $\mathcal{A}^{\mathrm{op}}_{\mathcal{D}} x \sim c$ ▷ *See Alg. 6.5.*
      return $\mathsf{AggregateQ}(x, c, op, \mathcal{D}, t, l, G)$;
    Case $\mathcal{C}^{\mathrm{op}}_{\mathcal{D}} \varphi \sim c$ ▷ *See Alg. 6.6.*
      return $\mathsf{CountingNeighboursQ}(\varphi, c, op, \mathcal{D}, t, l, G)$;
  end
end

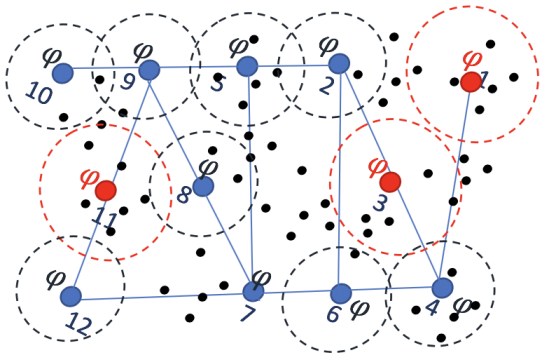


Figure 6.3: An example of city abstracted graph. A requirement is $\boxdot_{([0,+\infty),\mathsf{School})} \square_{[a,b]} (\mathcal{A}^{\mathsf{op}}_{([0,d],\mathsf{T})} \varphi \sim c)$ (The large nodes represent the locations of PoIs, among which the red ones represent the schools, and blue ones represent other PoIs. The small black nodes represent the locations of data sources.)

$$\mathsf{cost}(\varphi, l, G) = \begin{cases} 1 & \text{if } \varphi := p \vee \varphi := x \sim c \vee \varphi := \mathsf{True} \\ 1 + \mathsf{cost}(\varphi_1, l, G) & \text{if } \varphi := \neg\varphi_1 \\ \mathsf{cost}(\varphi_1, l, G) + \mathsf{cost}(\varphi_2, l, G) & \text{if } \varphi := \varphi_1 * \varphi_2, * \in \{\wedge, \mathcal{U}_I\} \\ |L^l_{\mathcal{D}}| & \text{if } \varphi := \mathcal{A}^{\mathsf{op}}_{\mathcal{D}} x \sim c \\ |L^l_{\mathcal{D}}| \mathsf{cost}(\varphi_1, l, G) & \text{if } \varphi := \mathcal{C}^{\mathsf{op}}_{\mathcal{D}} \varphi_1 \sim c \end{cases}$$

Algorithm 6.5: AggregateQ$(x, op, \mathcal{D}, \omega, t, l, G)$

```
begin
    Real v := 0; n := 0;
    if op == "min" then v := ∞ ;
    if op == "max" then v := −∞ ;
    L^l_D := deScan(l, G, D)
    for l' ∈ L^l_D do
        if op ∈ {min, max, sum} then
            v := op(v, π_x(ω)[t, l']);
        end
        if op == "avg" then
            v := sum(v, π_x(ω)[t, l']);
        end
        n := n + 1
    end
    if n == 0 then return ∞;
    if op == "avg" ∧ n ≠ 0 then return v/n − c ;
    if op == "sum" ∧ n ≠ 0 then return (v − c)/n ;
    else return v − c;
end
```

Algorithm 6.6: CountingNeighboursQ$(x, op, \mathcal{D}, \omega, t, l, G)$

```
begin
    Real n := 0, List s := Null;
    L^l_D := deScan(l, G, D)
    for l' ∈ L^l_D do
        s.add(Monitor(φ, ω, t, l', G))
        n := n + 1
    end
    if n == 0 then return ∞;
    else
        switch op do
            Case max
                return s.max()
            Case min
                return s.min()
            Case sum
                return s.max(round(c))
            Case avg
                return s.max(round(c × n))
        end
    end
end
```

Using the above function, the cost of each operation is calculated before "switch φ" (refer to alg:sastlQuanti). The cost function measures how complex it is to monitor a particular SaSTL formula. This can be used when the algorithm evaluates the $\wedge$ operator and it establishes the order in which the sub-formulas should be evaluated. The simpler sub-formula is the first to be monitored, while the more complex one is monitored only when the other sub-formula is satisfied. We update $\mathsf{monitor}(\varphi_1 \wedge \varphi_2, \omega)$ in Algorithm 6.7. With this cost function, the time complexity of the monitoring algorithm is reduced to $O(|\phi| \times T_{max} \times (log(n) + |L'|_{max}))$, where $|L'|$ is the maximal number of locations that an operation is executed with the improved parsing method. The improvement is significant for city requirements, where $|L'|_{max} < 100 \times |L|_{max}$.

Algorithm 6.7: Satisfaction of $(\varphi_1 \wedge \varphi_2, \omega)$

```
case φ1 ∧ φ2 do
    return Monitor(φ1, ω, t, l, G) ∧ Monitor(φ2, ω, t, l, G);
    if cost(φ1, l, G) ≤ cost(φ2, l, G) then
        if ¬ Monitor(φ1, ω, t, l, G) then
            return Monitor(φ2, ω, t, l, G);
        end
        return True;
    end
    if ¬ Monitor(φ2, ω, t, l, G) then
        return Monitor(φ1, ω, t, l, G);
    end
    return True;
end
```

6.4.3 Parallelization

In traditional STL monitor algorithm, the signals are checked sequentially. For example, to see if the data streams from all locations satisfy $\circledcirc_{\mathcal{D}} \Box_{[a,b]} \varphi$ in Figure 6.3, usually, it would first check the signal from location 1 with $\Box_{[a,b]}\varphi$, then location 2, and so on. At last, it calculates the result from all locations with $\circledcirc_{\mathcal{D}}$. In this example, checking all locations sequentially is the most time-consuming part, and it could reach over 100 locations in the field.

To reduce the computing time, we parallelize the monitoring algorithm in the spatial domain. To briefly explain the idea: instead of calculating a sub-formula ($\Box_{[a,b]}\varphi$) at all locations sequentially, we distribute the tasks of monitoring independent locations to different threads and check them in parallel. (Algorithm 6.8 presents the parallel version of the spatial counting operator $\mathcal{C}_{\mathcal{D}}$.) To start with, all satisfied locations $l' \in L^l_{\mathcal{D}}$ are added to a task pool (a queue). In the mapping process, each thread retrieves monitoring tasks (i.e., for $l_i, \Box_{[a,b]}\varphi$) from the queue and executes them in parallel. All threads only execute one task at one time and is assigned a new one from the pool when it finishes the last one, until all tasks are executed. Each task obtains the satisfaction of $\mathsf{Monitor}(\varphi, \omega, t, l, G)$ function, and calculates the local result v_i of operation $\mathsf{op}()$. The reduce step sums all the parallel results and calculates a final result of $\mathsf{op}()$.

Lemma 6.2. *The time complexity of the parallelized algorithm Monitor(ϕ, ω) is upper bounded by* $O(|\phi| T_{max}(log(n) + \frac{|L|_{max}}{P}))$ *when distributed to P threads.*

In general, the parallel monitor on the spatial domain reduces the computational time significantly. It is very helpful to support runtime monitoring and decision making, especially for a large number of requirements to be monitored in a short time. In practice, the computing time also depends on the complexity of temporal and spatial domains as well as the amount of data to be monitored. A

Algorithm 6.8: Parallelization of Counting of $(x, op, \mathcal{D}, \omega, t, l, G)$

Function `CountingNeighbours`$(\varphi, op, \mathcal{D}, \omega, t, l, G)$**:**

```
begin
    paratasks = Queue();
    for l' ∈ L^l_D do
        paratasks.add(l);
    end
    results = Queue();
    for i in 1..NumThreads do
        Thread_i ← worker(φ, ω, t, G);
    end
    Wait();
    return op(results);
end
```

Function `worker` (φ, ω, t, G)**:**

```
begin
    Real v := 0;
    if op == "min" then v := ∞; ;
    if op == "max" then v := −∞; ;
    while Num(tasks)>0 do
        l = paratasks.pop();
        moni = Monitor(φ, ω, t, l, G);
        v = op(v, moni);
    end
    results.add(v)
end
```

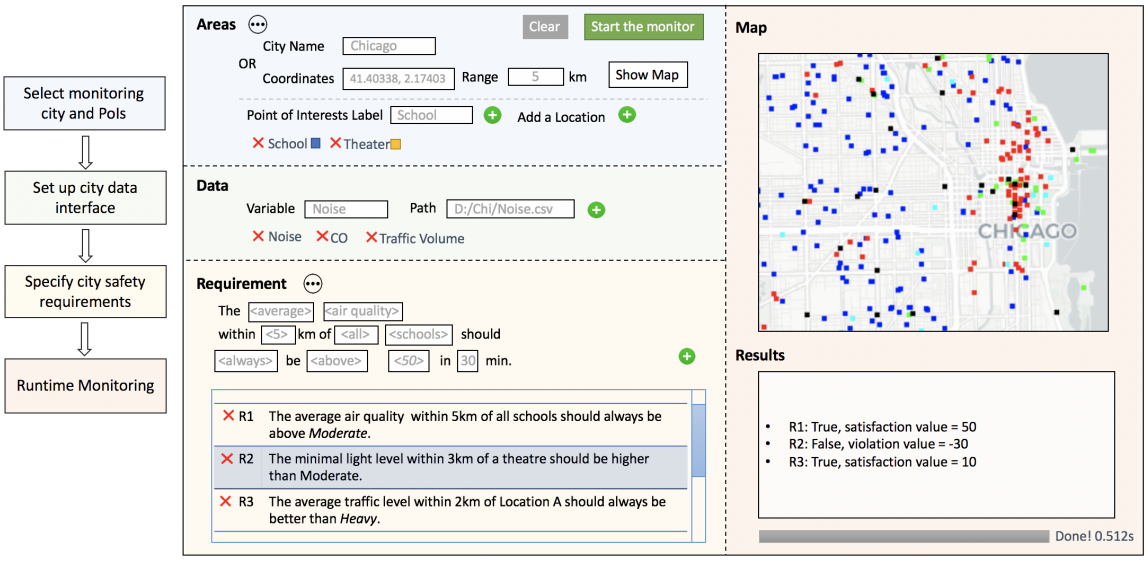


Figure 6.4: Interface of the SaSTL monitoring tool

comprehensive experimental analysis of the time complexity is presented in Section 6.7.

6.5 Tool for the SaSTL Monitor

We develop a user-friendly prototype tool for the SaSTL monitor that can support decision making of different stakeholders in smart cities. The interface and flowchart of the tool are shown in Figure 6.4. The tool allows users (e.g., city decision maker, citizens) without any formal method background to check the city performance (data) with their own requirements easily in four steps.

Step 1: selecting the monitoring city and PoI. To start with, users select the areas (such as a city, or a particular area of the city) to monitor, then choose the important labels that a requirement is involved with, such as, schools, parks, theaters, etc. Once selected, the important points of interest

(PoIs) are shown on the map. This helps users define and verify the monitoring locations. If a location or label is not included, users are also able to add them with their GPS coordinates. The map displays the locations of the specified labels and sensors. Users can enlarge the map to check the distribution of sensors and PoIs and revise the requirements accordingly.

Step 2: setting up the city data interface. The data of the city states collected from sensors across temporal and spatial domains are introduced to the monitor in the Data section. For the offline monitoring, users can specify the data location of each variable on the computer. For runtime monitoring, the sensing data continuously come into the computer, the data interface of which can be set up in this section.

Step 3: specifying the city safety requirements. As the next important step, users specify all requirements in the requirement section. Users first select the template and then choose/fill in the essential part using the structured template language. To be noted, the entities and spatial ranges correspond to the available data variables and PoIs inputs from the areas and data sections.

We define a series of templates using structured language learning from the existing city requirements, as shown in Figure 6.5. The goal of these templates is to help and inspire users to specify requirements precisely. These templates are adequate to represent all the example requirements given in Table 6.1 as well as the total set of 1,000 quantitatively-defined requirements. We define the templates in a recursive way. T is a template, and T1 and T2 are instances of T. The elements in T are optional, i.e. $< >$ can be defined as blank, indicating this element is not applicable or default in this requirement. For example, an environmental requirement is written as, "The <average> <air quality> within <1> mile of all <parks> should <always> be <above> <good>." The duration is interpreted as always (default) and there is no condition element. To convert a structured requirement to SaSTL, we extract the pre-defined key elements and translate them to the SaSTL formula following the rules. Meanwhile, users are also able to use the advanced features to input the city requirements in the format of the SaSTL formal formulas directly.

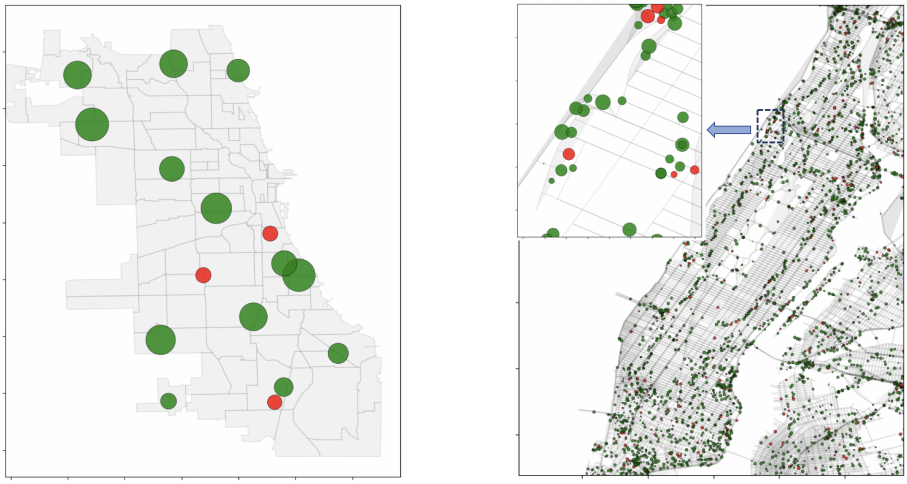
T:= The <aggregation operator> <entity> within <d> miles (from <a>th mile to <b>th mile) of <spatial operator> <PoIs> should <temporal operator> be <compare> <parameter> within <t> hours (from <m>th hour to <n>th hour / on <date> day).

T:= If **T1**, then **T2**.

T:= It is prohibited that **T1**.

T:= **T1** and/until/except **T2**.

Figure 6.5: Templates to specify city requirements



(1) Air Quality in Chicago (2) Traffic in New York City

Figure 6.6: Display of the Monitoring Results on the Maps (The green circle represents the location satisfied the requirement and the red circle represents the location violates the requirement; the size of the circle represents the degree of satisfaction or violation.)

Step 4: runtime monitoring. With all the data and requirements well defined, users can start the monitor in order to check if the incoming data from the smart city satisfies the requirements. The results are displayed with a Boolean value indicating if the requirement is satisfied and a robustness value indicating how much the requirement is satisfied or violated. In addition, the map also displays the monitor results visually. Two examples are shown in Figure 6.6. The first one is monitoring an air quality requirements of high schools in Chicago, and the second one is monitoring a traffic requirement in New York City. The green circle represents the location satisfied the requirement and the red circle represents the location violates the requirement; the size of the circle represents the degree of satisfaction or violation. Users can zoom in and out the map to focus on a specific area or check the overall performance as needed (See Figure 6.6 (2)).

In summary, we defined templates helping users to specify requirements to the SaSTL formal formulae. We believe these templates can not only help users to convert the requirement from English to formal formulae, they are also helpful for users to write the requirements much more specifically and precisely. The templates defined in this chapter are not sufficient to cover all the city requirements, especially the new requirements coming with more and more smart services being developed. However, the approach that using structured language to specify requirements proposed in this chapter is general and effective. Also, the templates are easily extended to adapt to new requirements.

We envision this tool can be used by different stakeholders, including but not limited to,

- *City managers and decision makers*: In the city operating center, with city data collected in real time, the Tool is able to help city managers and decision makers to monitor the data at runtime. It also helps the city center to detect conflicts, and provide support for decision makers by showing the trade-offs of satisfaction degrees among potential solutions.

- *City planners*: City planners, either from the government to make long-term policies or from a company to make a short-term event plan, they are able to use the Tool to verify the past city data with their requirements and make plans to prevent the violations.

- *Service designers*: Smart services are designed by different stakeholders including the government, companies and private parties, they are not aware of all the other services. However, with the monitor, they can test the influence of their services on the city and adjust the services to better serve the city.

- *Everyday citizens*: The tool can also provide a service to the everyday citizens. Citizens without any technical background are able to specify their own requirements and check them with the city data to find out in which areas of the city and period of the day their requirements are satisfied, and make plans about their daily life. For example, a citizen can specify an environmental requirement with his/her preferred air quality index and traffic conditions, and check the city data with the requirements and make up travelling agenda accordingly.

The use of the tool and surveys of stakeholders on its use is beyond the scope of this dissertation, but it could be accomplished in the future if it is used in a real city.

6.6 Coverage Analysis

We compare the specification coverage on 1000 quantitatively-specified real city requirements between STL, SSTL, STREL and SaSTL. The study is conducted by graduate students following the rules that if the language is able to specify the whole requirement directly with one single formula, then it is identified as True. To be noted, another spatial STL, SpaTeL is not considered as a baseline here, because it is not applicable to most of city spatial requirements. SpaTeL is built on a quad tree, and able to specify directions rather than the distance.

As shown in Figure 6.7, STL is only able to specify 184 out of 1000 requirements, while SSTL and STREL are able to formalize 431 requirements. SaSTL is able to specify 950 out of 1000 requirements.

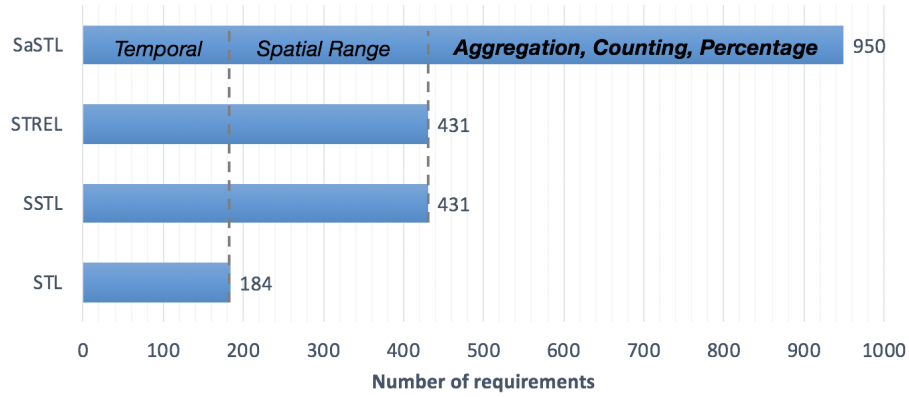
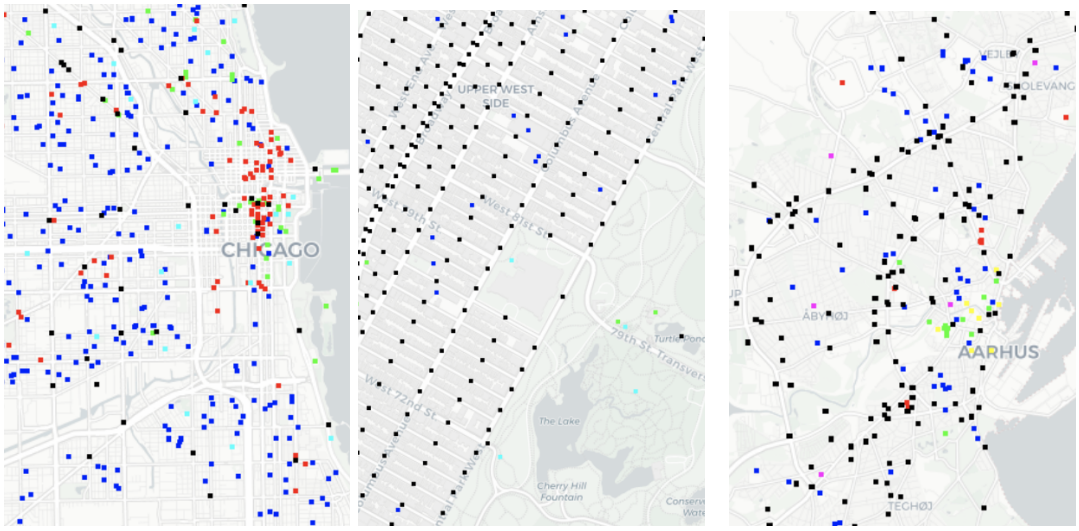


Figure 6.7: Comparison of the Specification Coverage on 1000 Real City Requirements

In particular, we made the following observations from the results. First, 50 requirements cannot be specified using any of the four languages because they are defined by complex math formulas that are ambiguous with missing key elements, relevant to the operations of many variables, or referring to a set of other requirements, e.g. "follow all the requirements from Section 201.12", etc. Secondly, SSTL, STREL and SaSTL outperformed STL in terms of requirements with spatial ranges, such as "one-mile radius around the entire facility"; Third, SSTL and STREL have the same coverage on the requirements that only contain a temporal and spatial range. Comparing to SSTL and SaSTL, STREL can also be applied to dynamic graph and check requirements reachability, which is very useful in applications like wireless sensor networks, but not common in smart city requirements; Fourth, the rest of the requirements (467 out of 1000) measure the aggregation of a set of locations, which can only be specified using SaSTL.

6.7 Evaluation

We evaluate the SaSTL monitor by applying it to three big city application scenarios, *New York*, *Chicago*, and *Aarhus*. We provide the information of three application scenarios in Table 6.3, including the area, data information (type, source, number of sensor nodes, time period and sampling rates), domains, monitoring variables, smart services, and evaluation metrics and baselines. Figure 6.8 presents the partial maps of three cities, where the locations of PoIs and sensors are marked. The experiments are evaluated on a server machine with 20 CPUs, each core is 2.2GHz, and 4 Nvidia GeForce RTX 2080Ti GPUs. The operating system is Centos 7.



(1) Chicago (2) Aarhus (3) New York

Figure 6.8: Partial Maps of Chicago, Aarhus and New York with PoIs and sensors annotated. (The black nodes represent the locations of sensors, red nodes represent the locations of hospitals, dark blue nodes represent schools, light blue nodes represent parks and green nodes represent theaters.)

Table 6.3: Information of Three Application Scenarios

	Chicago	**New York**	**Aarhus**
Area (km^2)	606	60	91.1
Data Type	Real-time States	Real-time Predicted States	Real-time States
Data Sources	Real Sensors	Simulated Sensors and Actuators	
Number of Locations	118	10,000	499
Time Period	2017.01-2019.05	-	2014.8-2014.10
Sampling Rate	1 min	10 seconds	1min
Application Domain	Environment, Public Safety	Environment, Transportation, Events, Emergencies, Public Safety	Environment, Transportation, Events
Variables	CO, NO, O_3, Visible light, Crime Rate	CO, NO, O_3, PM_x, Noise, Traffic, Pedestrian, Signal Lights, Emergency Vehicles, Accidents	Traffic, pollution, weather, parking, cultural event, library events

6.7.1 Runtime Monitoring of Real-Time Requirements in Chicago

We apply SaSTL to monitor the real-time requirements in Chicago. The framework is the same as shown in Figure 6.1, where we first formalize the city requirements to SaSTL formulas and then monitor the city states with the formalized requirements. Chicago is collecting and publishing city environment data (e.g., CO, NO, O3, visible light) every day since January, 2017 [110]. In our evaluation, we emulate the Chicago data as it arrives in real time, i.e. assuming the city was operating with our SaSTL monitor. Specifically, we monitor data from 118 locations between January, 2017 and May, 2019. In addition, we incorporate the Chicago crime rate data published by the city of Chicago [130]. The sampling rates of sensors vary by locations and variables (e.g., CO is updated

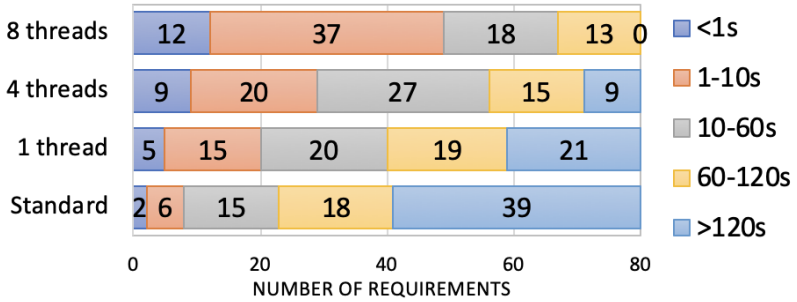


Figure 6.9: Number of Requirements Checked on Different Computing Time (x: the number of requirements, y: the number of threads, bars: different periods of computing time)

every few seconds, and the crime rate map is updated by events), so we normalize the data frequency as one minute. Then we specify 80 safety and performance requirements that are generated from the real requirements, and apply the SaSTL to monitor the data every 3 hours continuously to identify the requirement violations.

Table 6.4: Safety and Performance Requirements for Chicago

	Requirement	**SaSTL**
R1	The average air quality within 5km of all schools should always be above *Moderate* in the next 3 hours.	$\boxdot_{([0,+\infty),\text{School})} \Box_{[0,3]} (\mathcal{A}^{\text{avg}}_{([0,5],\top)} x_{\text{air}} > \text{Moderate})$
R2	The worst air quality all over the city should be better than *Very Unhealthy* all time.	$\mathcal{A}^{\text{max}}_{\mathcal{D}} x_{\text{air}} > \text{VeryUnhealthy}$
R3	The average air quality within 3 km of the park should be better than *Good*.	$\boxdot_{\mathcal{D}} \Box_{[0,3]} (\mathcal{A}^{\text{avg}}_{[0,3]} x_{\text{air}} > \text{Good}) \wedge \text{Park}$
R4	If the weather is rainy (average humidity within 3 km of a park >50), then the average air quality within 3 km of the park should be better than *Unhealthy for Sensitive Groups*.	$\boxdot_{\mathcal{D}} \Box_{[0,3]} (\mathcal{A}^{\text{avg}}_{[0,3]} x_{\text{humidity}} > 0.5 \rightarrow \mathcal{A}^{\text{avg}}_{[0,3]} x_{\text{air}} > 3) \wedge \text{Park}$
R5	The minimal light level should be larger than *Moderate* within 3km of a theatre.	$\boxdot_{\mathcal{D}} \Box_{[0,3]} (\mathcal{A}^{\text{min}}_{[0,3]} x_{\text{light}} > \text{Moderate}) \wedge \text{Theatre}$
R6	For the blocks with a high crime rate, the average light level within 3 km should always be *High*.	$\boxdot_{([0,+\infty),\top)} \Box_{[0,3]} (x_{\text{Crime}} >= \text{High} \rightarrow \mathcal{A}^{\text{avg}}_{([0,3],\top)} x_{\text{Light}} >= \text{High})$

Valuable information is identified from the monitor results of different periods during a day. We randomly select 30 days of weekdays and 30 days of weekends. We divide the daytime of a day into 4 time periods and 3 hours per time period. We calculate the percentage of satisfaction (i.e., number of satisfied requirement days divides 30 days) for each time period, respectively. The results of six example requirements R1 to R6 are shown in Figure 6.10. The SaSTL monitor results can be potentially used by different stakeholders.

First, *with proper requirements defined, the city decision makers are able to identify the real problems and take actions to resolve or even avoid the violations in time.* For example, we could see over 20% of the time the requirements are missed everyday. Based on the monitoring results of requirement

Figure 6.10: Evaluation for Chicago

R1, decision makers can take actions to redirect the traffic near schools and parks to improve the air quality. Another example of requirement R6, the satisfaction is much higher (up to 33% higher in R6, 8pm - 11pm) over weekends than workdays. There are more people and vehicles on the street on weekends, which as a result also increases the lighted areas. However, as shown in the figure, the city lighting in the areas with high crime rate is only 60%. An outcome of this result for city managers is that they should pay attention to the illumination of workdays or the areas without enough light to enhance public safety.

Second, *it gives the citizens the ability to learn the city conditions and map that to their own requirements*. They can make decisions on their daily living, such as the good time to visit a park. For example, requirement R1, 11am - 2pm has the lowest satisfaction rate of the day. The instantaneous air quality seems to be fine during rush hour, but it has an accumulative result that affects citizens' (especially students and elderly people) health. A potential suggestion for citizens who visit or exercise in the park is to avoid 11am - 2pm. Additionally, the satisfaction rates are more time sensitive for a local area, like a school and park, but tend to be similar within the day for the overall city (R2). Furthermore, the best air quality over the city is in the evening of the weekends, which is a good time for citizens to take a walk outdoors.

We count the average monitoring time taken by each requirement when monitoring for 3-hour data. Then, we divide the computing time into 5 categories, i.e., less than 1 second, 1 to 10 seconds, 10 to

Table 6.5: Safety and Performance Requirements for New York City

	Requirement	SaSTL
NYR1	The average noise level in the school area (within 1km) should always be less than 50dB in the next 30min.	$\boxdot_{([0,+\infty),\mathsf{School})} \Box_{[0,30]} (\mathcal{A}^{\mathrm{avg}}_{([0,1],\tau)} x_{\mathsf{Noise}} < 50)$
NYR2	If an accident happens, at least one of the nearby hospitals (within 5km), its traffic condition within 2km should not reach the level of congestion in the next 60 min.	$\boxdot_{([0,+\infty),\tau)}(\mathsf{Accident} \rightarrow \mathcal{C}_{([0,5],\mathsf{Hospital})}(\Box_{[0,60]}(\mathcal{A}^{\mathrm{avg}}_{([0,2],\tau)} x < \mathsf{Congestion})) > 0)$
NYR3	If there is an event, the max number of pedestrians waiting at an intersection should not be greater than 50 for more than 10 minutes.	$\boxdot_{([0,+\infty),\tau)}(\mathsf{Event} \rightarrow \Box_{[0,10]}(\mathcal{A}^{\max}_{([0,1],\tau)} x_{\mathsf{ped}} < 50))$
NYR4	At least 90% of the streets, the PMx emission should not exceed *Moderate* in 60 min.	$\mathcal{C}^{\mathrm{avg}}_{([0,+\infty),\tau)}(\Box_{[0,60]}(\mathcal{A}^{\max}_{([0,1],\tau)} x_{\mathsf{PMx}} < \mathsf{Moderate})) > 0.9$
NYR5	If an accident happens, it should be solved within 60 min, and before that nearby (500 m) traffic should be above moderate on average and safe in worst case.	$\boxdot_{([0,+\infty),\tau)}(\mathsf{Accident} \rightarrow (\mathcal{A}^{\mathrm{avg}}_{([0,500],\tau)} x_{\mathsf{traffic}} < \mathsf{Moderate} \wedge \mathcal{A}^{\max}_{([0,500],\tau)} x_{\mathsf{traffic}} < \mathsf{Safe}) \mathcal{U}_{[0,60]} \neg \mathsf{Accident})$

60 seconds, 60 to 120 seconds, and longer than 120 seconds, and count the number of requirements under each category under the conditions of standard parsing, improved parsing with single thread, 4 threads, and 8 threads. The results are shown in Figure 6.9. Comparing the 1st (standard parsing) and 4th (8 threads) bar, without the improved monitoring algorithms, for about 50% of the requirements, each one takes more than 2 minutes to execute. The total time of monitoring all 80 requirements is about 2 hours, which means that the city decision maker can only take actions to resolve the violation at earliest 5 hours later. However, with the improved monitoring algorithms, for 49 out of 80 requirements, each one of them is executed within 60 seconds, and each one of the rest requirements is executed within 120 seconds. The total execution time is reduced to 30 minutes, which is a reasonable time to handle as many as 80 requirements. More importantly, it illustrates the effectiveness of the parsing function and parallelization methods. Even if there are more requirements to be monitored in a real city, it is doable with our approach by increasing the number of processors.

6.7.2 Runtime Conflict Detection and Resolution in Simulated New York

The framework of runtime conflict detection and resolution [131, 38] considers a scenario where smart services send action requests to the city center, and where a simulator predicts how the requested actions change the current city states over a finite future horizon of time. Then it checks the predicted states against city requirements. If the requirements are satisfied, the requested actions will be approved to execute in the city. If there exists a requirement violation within the future horizon, a conflict is detected. CityResolver will be applied to resolve the conflicts. Details of the resolution are not the main part of this chapter, please refer to the previous chapter. Note that with the conflicts

detected and resolved, the city's future states will be affected. In this paper, we apply the SaSTL monitor to specify requirements with spatial aggregation and check the *predicted spatial-temporal data* with the SaSTL formulas.

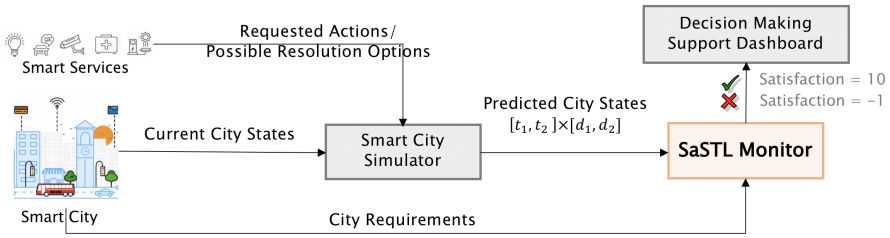


Figure 6.11: Applying the SaSTL Monitor to the Conflict Detection and Resolution Architecture in New York City

We set up a smart city simulation of New York City using the Simulation of Urban MObility (SUMO) [99] with the traffic pattern (vehicle in-coming rate of key streets) from real city data [100], on top of which, we implement 10 services (S1: Traffic Service, S2: Emergency Service, S3: Accident Service, S4: Infrastructure Service, S5: Pedestrian Service, S6: Air Pollution Control Service, S7: PM2.5/PM10 Service, S8: Parking Service, S9: Noise Control Service, and S10: Event Service). The real-time states (including CO, NO, O3, PMx, Noise, Traffic, Pedestrian Number, Signal Lights, Emergency Vehicles, and Accident number) from the domains of environment, transportation, events and emergencies are obtained from about 10,000 simulated nodes. Then, we apply the STL Monitor as the *baseline* to compare the capability of requirement specification and the ability to improve city performance. We simulate the city running for 30 days with sampling rate as 10 seconds in two control sets, one without any monitor and one with the SaSTL monitor. For the first set (no monitor), there is no requirement monitor implemented. For the second one (SaSTL monitor), five examples of different types of real-time requirements and their formalized SaSTL formulas are given in Table 6.5.

The results are shown in Table 6.6. We measure the city performance from the domains of transportation, environment, emergency and public safety using the following metrics, the total number of violations detected (i.e., the total number of safety requirements violated during the whole simulation time), the average CO (mg) emission per street, the average noise (dB) level per street, the emergency vehicles waiting time per vehicle per intersection, the average number and waiting time of vehicles waiting in an intersection per street, and the average pedestrian waiting time per intersection.

We make some observations by comparing and analyzing the monitoring results.

Table 6.6: Comparison of the City Performance with the STL Monitor and the SaSTL Monitor

	No Monitor	**SaSTL Monitor**
Number of Violation	Unknown	**173**
Air Quality Index	67.91	**40.18**
Noise (db)	73.32	**41.42**
Emergency Waiting Time (s)	20.32	**11.88**
Vehicle Waiting Number	22.7	**12.6**
Pedestrian Waiting Time (s)	190.2	**61.1**
Vehicle Waiting Time (s)	112.12	**59.22**

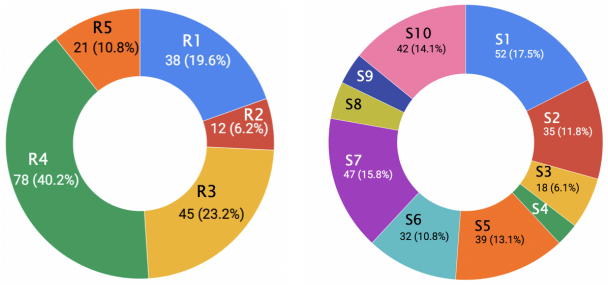
Table 6.7: Computing time of requirements with standard parsing function, with improved parsing functions and different number of threads

	Standard Parsing (s)	**1 thread (s)**	**4 threads (s)**	**8 threads (s)**
NYR1	2102.13	140.29	50.31	**26.12**
NYR2	55.2	**0.837**	1.023	0.912
NYR3	69.22	22.25	7.54	**4.822**
NYR4	390.19	390.19	100.23	**53.32**
NYR5	61.76	61.76	20.25	**15.68**
Total	2678.5	615.32	179.35	**100.85**

First, *the SaSTL monitor obtains a better city performance with fewer number of violations detected under the same scenario.* As shown in Table 6.6, on average, the framework of conflict detection and resolution with the SaSTL monitor improves the air quality by 40.8%, and improves the pedestrian waiting time by 47.2% comparing to the one without a monitor.

Second, *the SaSTL monitor reveals the real city issues, helps refine the safety requirements in real time and supports improving the design of smart services.* We also compare the number of violations on each requirement. The results (Figure 6.12 (1)) help the city managers to measure city's performance with smart services for different aspects, and also help policymakers to see if the requirements are too strict to be satisfied by the city and make a more realistic requirement if necessary. For example, in our 30 days simulation, apparently, NYR4 on air pollution is the one requirement that is violated by most of the smart services. Similarly, Figure 6.12 (2) shows the number of violations caused by different smart services. Most of the violations are caused by S1, S5, S6, S7, and S10. The five major services in total cause 71.3% of the violations. City service developers can also learn from these statistics to adjust the requested actions, the inner logic and parameters of the functions of the services, so that they can design a more compatible service with more acceptable actions in the city.

We compare the average computing time for each requirement under four conditions, (1) using the standard parsing algorithm without the cost function, (2) improved parsing algorithm with a single thread, (3) improved parsing algorithm with spatial parallelization using 4 threads and (4) using 8 threads. The results are shown in Table 6.7.



(1) Requirements (2) Smart Services

Figure 6.12: Distributions of the violations over requirements and smart services

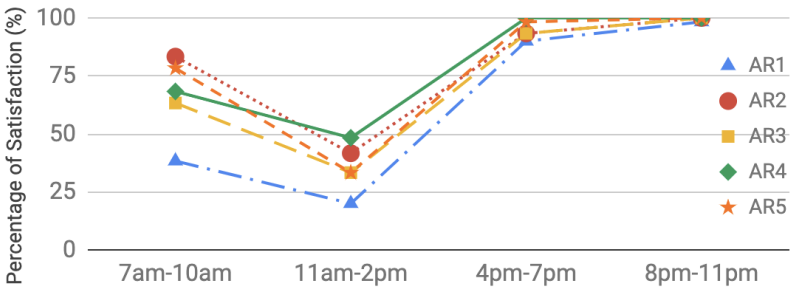


Figure 6.13: Comparisons of Satisfaction Rate on AR1 to AR5

From the results we observe that, first, the improved parsing algorithm reduces the computing time significantly for the requirement specified on PoIs, especially for NYR1 that computing time reduces from 2102.13 seconds to 140.29 seconds (about 15 times). Second, the parallelization over spatial operator further reduces the computing time in most of the cases. For example, for NYR1, the computing time is reduced to 26.12 seconds with 8 threads while 140.29 seconds with single thread (about 5 times). When the amount of data is very small (NYR2), the parallelization time is similar to the single thread time, but still much more efficient than the standard parsing.

The results demonstrate the effectiveness and importance of the efficient monitoring algorithms. The total time of monitoring 5 requirements is reduced from 2678.5 seconds to 100.85 seconds. In the real world, when multiple requirements are monitored simultaneously, the improvement is extremely important for real-time monitoring.

6.7.3 Evaluation for Aarhus

In this case study, we monitor the past data of events and states from Aarhus to show how the SaSTL monitor helps to understand the effects caused by events and therefore aids in decision making for city events. We utilize 60 days (August to September 2014) of Aarhus city data collected simultaneously across the domains of transportation (e.g., traffic volume, parking), events (e.g., cultural events and

library events) and the environment (generated pollution and weather). All the data were collected from 449 observation points and published by CityPulse [132]. Data was collected with different sampling rates (e.g., the traffic data were aggregated by 5 minutes and events data were recorded by the event time), thus for the monitoring purpose, we normalize the data frequency as 5 minutes. Five safety and performance requirements and their corresponding SaSTL statements are presented with a high demand for aggregations specified for Aarhus in Table 6.8. Basically, AR1 to AR5 specify that when there is an event, there is a different level of safety requirements on the traffic under different circumstances. For example, AR2 focuses on the areas nearby an event, AR3 focuses on the safety of school with an event, and R4 considers the effects from extreme weather conditions. AR5 has a big picture on all schools across the city when a large cultural event is happening.

Table 6.8: Safety and Performance Requirements for Aarhus

	Requirement	**SaSTL**
AR1	If there is an event, the traffic level nearby should always be better than *Moderate*.	$\mathsf{Event} \rightarrow \boxdot_{\mathcal{D}} \Box_{[0,3]}\, x_{\mathrm{traffic}} > \mathsf{Moderate}$
AR2	If there is an event, the average traffic level nearby should always be better than *Moderate* and the maximum traffic level nearby should be better than *Safe*.	$\mathsf{Event} \rightarrow \boxdot_{\mathcal{D}} \Box_{[0,3]}\, (\mathcal{A}^{\mathrm{avg}}_{([0,1],\top)} x_{\mathsf{traffic}} > \mathsf{Moderate} \wedge \mathcal{A}^{\mathrm{max}}_{([0,1],\top)} x_{\mathsf{traffic}} > \mathsf{Safe})$
AR3	If there is an event, the average traffic near the school (3km) should always be better than *Moderate* and the maximum traffic level should be better than *Heavy*.	$\mathsf{Event} \rightarrow \boxdot_{\mathcal{D}} \Box_{[0,3]}\, (\mathcal{A}^{\mathrm{avg}}_{([0,1],\top)} x_{\mathsf{traffic}} > \mathsf{Moderate} \wedge \mathcal{A}^{\mathrm{max}}_{([0,1],\top)} x_{\mathsf{traffic}} > \mathsf{Heavy}) \wedge \mathsf{School}$
AR4	If there is an event and the weather is rainy or snowy heavily, the average traffic level around school should be better than *Heavy*	$\mathsf{Event} \wedge \mathsf{Humidity} > 50\% \rightarrow \boxdot_{\mathcal{D}} \Box_{[0,3]}\, (\mathcal{A}^{\mathrm{avg}}_{([0,1],\top)} x_{\mathsf{traffic}} > \mathsf{Heavy}) \wedge \mathsf{School}$
AR5	With big cultural events going on the city, over the city, 80% schools' average traffic volume nearby (3km) should always be better than *Moderate*.	$\mathsf{Event} \rightarrow \mathcal{C}_{\mathcal{D}} \Box_{[0,3]}\, (\mathcal{A}^{\mathrm{avg}}_{([0,1],\top)} x_{\mathsf{traffic}} > \mathsf{Moderate}) \wedge \mathsf{School} > 80\%$

The monitoring results from Aarhus are shown in Figure 6.13. The percentage of satisfaction equals to the number of requirement satisfied days divided by 60 days. The following are observations on the requirements and monitoring results.

- Comparing the monitoring results on AR1 and AR2, AR1 has a much lower satisfaction rate. It also leads to a higher and reliable satisfaction rate.

- Comparing to AR2, for the same events, AR3 moves its focus on the area nearby schools. The results, however, are lower than AR2. It means that events have more influence on the school areas, which should draw attention from the city managers. Students should reduce or avoid activities during this time when there is an event going on nearby.

- During 11am to 2pm, the overall performance on all five requirements are worst, even less than 50%. It is actually the time period right after a morning event or before an afternoon event.

The monitoring results help the city managers have a better view of the distribution of effects from events.

- We also find that the satisfaction rate is very high (almost 100%) after 8pm. The reasons for that are the schools are usually closed at that time, and most of cultural and library events happen during the day. In other cities or events, the distribution will be different. However, the SaSTL monitor is general enough to help citizens and managers detect it.

The evaluation on Aarhus shows how the SaSTL monitor helps the city to understand the effects on the city from events and make better plans for events. Usually, areas with an event get caught up in complicated situations, such as paralyzed traffic, long queues with a large amount of people, emergencies and accidents. Therefore, playing back and analyzing the city data during events is extremely important for cities to avoid emergency situations for future events.

6.8 Summary

In this chapter, we present a novel Spatial Aggregation Signal Temporal Logic to specify and to monitor requirements of smart cities at runtime. We develop an efficient monitoring framework that optimizes the requirement parsing process and can check in parallel a SaSTL requirement over multiple data streams generated from thousands of sensors that are typically spatially distributed over a smart city. SaSTL is a powerful specification language for smart cities because of its capability to monitor the city desirable features of temporal (e.g., interval), spatial (e.g., PoIs, range) and their complicated relations (e.g. always, everywhere, aggregation) between them. More importantly, it can coalesce many requirements into a single SaSTL formula and provide the aggregated results efficiently, which is a major advance on what smart cities do now. The development of 5G and 6G could better support the monitoring and communication among sensors, services and the city center. We believe it is a valuable step towards developing a practical smart city monitoring system even though there are still open issues for future work. Furthermore, SaSTL monitor can also be easily generalized and applied to monitor other large-scale IoT deployments at runtime efficiently. In the future, we will explore its capability to specify and monitor other properties and requirements (e.g., security and privacy).

Chapter 7

Logic Calibrated Uncertainty

Predictive monitoring concerns the problem of (continuously) making predictions about future states and monitoring if the predicted states satisfy or violate requirements. Predictive monitoring offers a promising paradigm in supporting the decision making of Cyber-Physical Systems (CPS), for example, reducing an automated insulin delivery system's dosage if a potentially dangerous hypoglycemic condition is predicted, and adapting a traffic control system's signaling if traffic congestion due to car accidents or inclement weather is forecast. On the one hand, various machine learning and statistical analysis techniques (e.g., neural networks, ARIMA) have been popularly applied to predict future states of CPS across different application domains, such as predicting glucose levels for artificial pancreas systems [133], predicting takeover reaction time for automated vehicles [134], forecasting air quality [103], fire risk [135], and frost damage [136] in smart cities. On the other hand, there have been great efforts over the past decades devoted to develop runtime monitoring techniques and tools. For example, a survey of specification (e.g., Signal Temporal Logic (STL) [137]) based runtime monitoring of CPS is provided in [67]. Nevertheless, research on predictive monitoring that addresses challenges arisen from combining these two aspects has received scant attention until very recently. Existing works of predictive monitoring (e.g., [57, 59]) mostly focus on monitoring individual predictions rather than sequential predictions. A more recent work [66] considers STL-based monitoring for predictions made from statistical time-series analysis, assuming that a joint probability distribution of predictions over multiple time-points can be estimated.

In this chapter, we develop a novel approach for monitoring sequential predictions generated from

Bayesian Recurrent Neural Networks (RNN) models. RNN-based sequential prediction has been widely used in CPS applications (e.g., [138, 139]). Many commonly used RNN models (e.g., LSTM) are deterministic, which generate the same sequence of predictions given the same set of historical states. A key challenge is how to generate and monitor predictions that can capture the inherent uncertainty in CPS (e.g., due to sensing noise, human interactions). We study two real-world CPS datasets to analyze the uncertainty characteristics and implications on predictive monitoring. Insights from our study show that (i) deterministic RNN models are not suitable for representing the significant uncertainty exhibited in CPS, and (ii) there is a need for developing new monitors for checking sequential predictions with high uncertainty.

To address the first insight, we apply stochastic regularization techniques (SRTs) [80] to cast deterministic RNNs as Bayesian RNNs, which adapt deterministic sequential predictions as a sequence of posterior probability distributions to estimate the uncertainty. We formally define a *flowpipe* signal to represent uncertain sequential predictions generated by Bayesian RNNs. The projection of a flowpipe for a single time-point is a confidence interval induced from a Gaussian distribution, which includes values of all possible sequences predicted by the Bayesian RNN. A larger confidence interval indicates a higher level of uncertainty about the prediction.

To address the second insight, we propose a new logic named *Signal Temporal Logic with Uncertainty* (STL-U). Existing temporal logic based monitors (e.g., STL and its variants) mostly focus on deterministic signals and cannot be directly applied for monitoring an infinite set of sequences contained in a flowpipe. Several recent works (e.g., [69, 70, 71, 72]) extend STL with stochastic predicates to reason about uncertainty. Our approach differs from these previous works fundamentally. Instead of reasoning about the probability of satisfying a predicate, STL-U checks a flowpipe signal containing an infinite set of sequences. For example, consider a STL-U formula $\Box_{[0,2]}\mathsf{AQI}_{\varepsilon=95\%} < 50$, which represents the requirement "the predicted Air Quality Index under 95% confidence level should never exceed 50 in the next two hours". We develop a STL-U monitor that checks if all (*resp.* some) sequences contained in the predicated flowpipe satisfy the requirement, which we call STL-U strong (*resp.* weak) satisfaction. In addition, we equip the STL-U monitor with the capability to answer queries such as "Under what confidence level, the predicated flowpipe is guaranteed to strongly (weakly) satisfy the STL-U formula?" It is particularly useful to compute such confidence guarantees when users do not know *a priori* about the level of prediction uncertainty.

Furthermore, the quality of predictive monitoring results depends on the uncertainty estimated from

Bayesian RNNs, which varies based on the choice of uncertainty estimation schemas (e.g., SRTs and dropout rates). In the current practice, an uncertainty estimation schema is often selected empirically or guided by traditional deep learning metrics (e.g., mean square error, negative log-likelihood, KL divergence), which tend to over-estimate the uncertainty level [140, 82]. In addition, these metrics treat the uncertainty estimation of each individual value in a predicted sequence separately, and thus lack an integrated view about the uncertainty of the sequence. To address this limitation, we develop novel criteria that leverage STL-U monitoring results to select and tune uncertainty estimation schemas. Such STL-U criteria can help to calibrate the uncertainty estimates for predictive monitoring.

We compare STL-U criteria with state-of-the-art baselines via experimental evaluation on real-world CPS datasets. The results are very promising: STL-U criteria outperform all six baselines in terms of F1-scores comparing STL-U monitoring results for the predicted and target sequences. In addition, experiments also show that STL-U uncertainty calibration is compatible with different types of RNN models.

Finally, we evaluate the STL-U based predictive monitoring approach via a simulated smart city case study with 10 smart services and 390 requirements. Experiment results demonstrate the efficiency of the approach. In one case, it only takes around 281 seconds to monitor 130,000 predicted flowpipes. Moreover, our approach can better support decision making in the simulated smart city. The simulation results show that our approach improves various city performance metrics (e.g., emergency waiting time, vehicle waiting number) significantly when compared with two baselines.

Contributions. We summarize the major contributions of this chapter as follows.

- We develop a novel STL-U based predictive monitoring approach for CPS, which continuously monitors uncertain sequential predictions about future states generated by Bayesian RNN models.
- We create novel STL-U criteria for calibrating uncertainty estimation in Bayesian deep learning.
- We evaluate the proposed approach via real-world smart city datasets and a simulated smart city case study, which show encouraging results.

7.1 Motivating Study

In this section, we study the following real-world smart city datasets as motivating examples to analyze uncertainty characteristics and to discuss implications on predictive monitoring for CPS.

1. *Air quality dataset* [103] collected by Microsoft Research from 437 air quality monitoring stations in China during the period of 5/1/2014 to 4/30/2015, which includes 2,891,393 records of air quality index (AQI).

2. *Traffic volume dataset* [100] collected by the NYC Department of Transportation from 1,490 street segments in the New York City during the period of 9/13/2014 to 4/5/2018, which includes 514,776 records of traffic volume count.

Uncertainty characteristics. We made the following observations by analyzing these datasets.

- *Significant uncertainty exists in smart cities and the uncertainty level varies across different locations.* As an illustrative example, Figure 7.1 shows 10-hour data segments taken from three different stations in the air quality dataset. We preprocessed the raw data by averaging the data within an hour and performing a uniform quantization [141]. Figure 7.1 plots data segments with the same prefixes (i.e., the same average AQI levels) for the first five hours. However, these data segments show significant uncertainty in the suffixes. The light shadows in the figure cover the entire data range, and the dark shadows represent the range of 95% percentile [1] of the corresponding normal distribution at a time. A larger range of 95% percentile indicates a higher level of data uncertainty. Thus, Figure 7.1 shows that station 1 has the highest uncertainty level, followed by station 2 and station 3.

- *The data uncertainty level is affected by the pre-knowledge (i.e., the prefix length of data segments).* As an illustrative example, Figure 7.2 plots 10-hour data segments taken from the same location in the traffic volume dataset. We preprocessed the data by averaging the traffic volume counts within an hour and performing a logarithmic quantization [141]. Figure 7.2 shows that, as the length of common data segment prefixes increases (i.e., more pre-knowledge about the data), the uncertainty level reduces.

The uncertainty in CPS data could arise from many sources, such as noise from the sensing data (e.g., reading errors, faults, anomalies), the environment (e.g., unexpected weather or events like

[1] We utilize 95% percentile because it is commonly used to represent the majority of the population distributed [142].

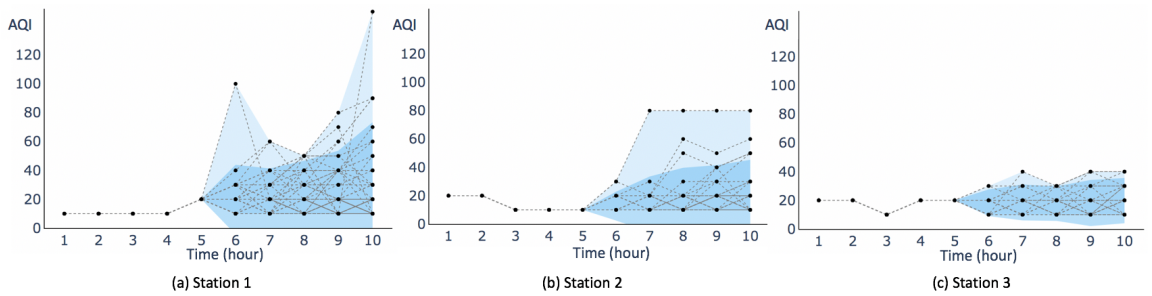


Figure 7.1: The uncertainty level varies across different stations in the air quality dataset.

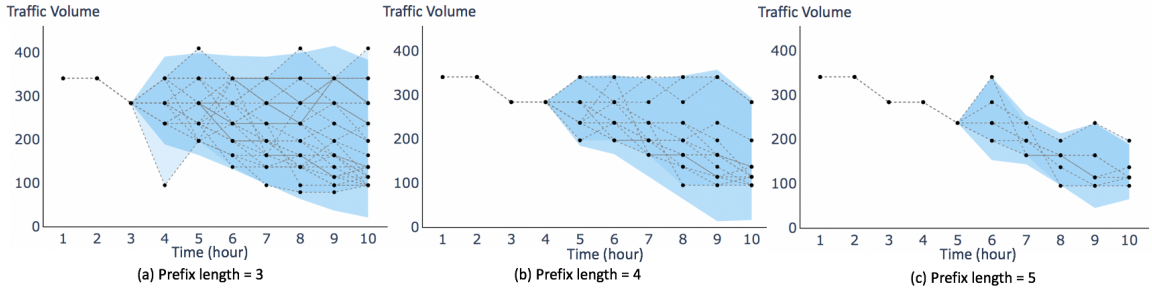


Figure 7.2: The uncertainty level varies for different pre-knowledge (prefix lengths) in the traffic volume dataset.

accidents), and human behaviors (e.g., interventions from human operators), to name a few. Thus, predictive monitoring for CPS should account for the impact of uncertainty.

Implications on predictive monitoring. We discuss how the uncertainty in CPS would affect predictive monitoring from two aspects: (i) prediction, and (ii) monitoring.

First, existing deterministic prediction models (e.g., RNNs) mostly forecast future states based on historical states. Given the same historical data, a deterministic model always yields the same prediction about future states. However, as discussed above, real-world CPS data exhibits significant uncertainty. For example, Figure 7.1 illustrates that data segments with the same average AQI levels for the first five hours can lead to a diverse range of trends for the following five hours. Thus, deterministic prediction models are not suitable to capture the uncertainty in CPS data. There is a need for developing new techniques that can predict future states with appropriate levels of uncertainty.

Second, existing works (e.g., [38]) have applied Signal Temporal Logic (STL) to specify and monitor city requirements. For example, a requirement that "the AQI level within 10 hours should always be below certain threshold λ" can be specified with a STL formula $\Box_{[0,10]}(\mathsf{AQI} < \lambda)$, where $\Box$ is the

Table 7.1: Number of satisfying data segments for each STL formula.

STL formulas	$\lambda = 50$	$\lambda = 51$	$\lambda = 70$	$\lambda = 75$	$\lambda = 80$
$\square_{[0,10]}\mathsf{AQI} < \lambda$	2,807	2,895	3,614	4,011	4,443
$\Diamond_{[0,10]}\mathsf{AQI} < \lambda$	6,670	6,731	7,304	7,408	7,493
$\mathsf{AQI} < 150\ \mathcal{U}_{[0,10]}\ \mathsf{AQI} < \lambda$	5,558	5,613	6,030	6,169	6,230
$\square_{[0,10]}\mathsf{Traffic} < \lambda$	1,241	1,359	3,090	3,332	3,532
$\Diamond_{[0,10]}\mathsf{Traffic} < \lambda$	4,220	4,283	4,546	4,563	4,598

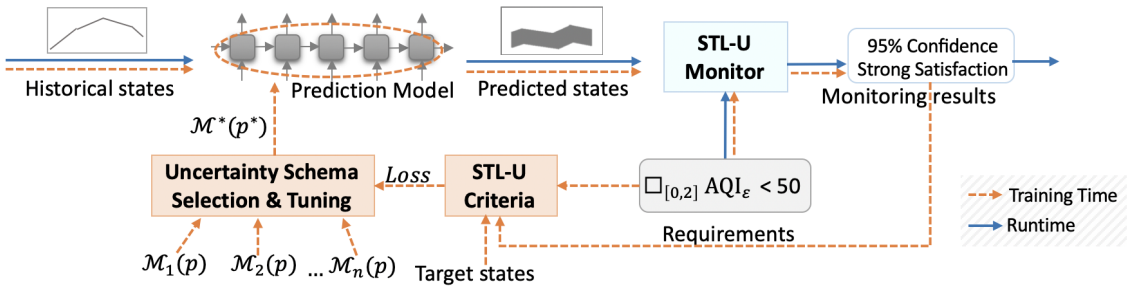


Figure 7.3: Overview of STL-U based predictive monitoring approach.

temporal logical operator representing "always" and λ is a parameter (e.g., $\lambda = 50$ for good air quality). Table 7.1 shows five example STL formulas, with the first three representing city requirements about AQI and the last two representing city requirements about traffic volume. We applied a STL monitor to check how many 10-hour data segments of a selected location in the air quality and traffic volume datasets satisfy these STL formulas with varying parameter values of λ. We observe from Table 7.1 that the STL monitoring results can be very sensitive to the change of λ values. For example, the number of satisfying data segments for $\Diamond_{[0,10]}(\mathsf{AQI} < \lambda)$ increases by 61 (from 6,670 to 6,731) when the λ value only increases by 1 (from 50 to 51), and goes up 85 (from 7,408 to 7,493) when the λ value increases from 75 to 80. Even though the differences also vary by the type of requirements, the amount is still too large to ignore. From the perspective of the data, it shows that a small difference of the data could completely change the monitoring results. However, it is impossible to predict the data with 100% accuracy due to the existence of uncertainty in CPS, which makes the monitoring results less effective to support decision making. It also indicates that the existing monitors are not suitable for data with high uncertainty. Therefore, there is a need for developing new monitors that can check the prediction results accounting for the uncertainty.

7.2 Approach Overview

We develop a novel predictive monitoring approach for CPS, as illustrated in Figure 7.3, to address limitations discussed in the previous section. Our approach adopts Bayesian RNN models to predict

future states (e.g., AQI in next 2 hours) based on historical data (e.g., AQI in the past 5 hours). By contrast to deterministic prediction models that output a single sequence of predicted values, Bayesian RNN models generate a sequence of distributions to capture the uncertainty of predicted future states, which are represented by a range of potential values under a given confidence level at each time-point. We propose a new logic named *Signal Temporal Logic with Uncertainty* (STL-U) and develop a STL-U monitor to check such uncertain sequential predictions generated by Bayesian RNN models. STL-U is expressive enough to specify CPS requirements with uncertainty confidence levels. For example, a STL-U formula $\Box_{[0,2]}\mathsf{AQI}_{\varepsilon=95\%} < 50$ represents the requirement "the predicted AQI under 95% confidence level should never exceed 50 in the next two hours". The STL-U monitor checks if all or some possible sequences of future states predicted by the Bayesian RNN model satisfy the requirement, which we call strong and weak satisfaction relations. When the confidence level is unspecified in a formula (e.g., $\Box_{[0,2]}\mathsf{AQI}_{\varepsilon=?} < 50$), we can also use STL-U monitor to compute the range of confidence levels under which the predicted flowpipe is guaranteed to strongly or weakly satisfy a requirement. In addition, we develop novel criteria (loss functions) based on STL-U monitoring results to calibrate the uncertainty estimation in Bayesian deep learning.

At *training* time (the flow marked by orange dash-lines in Figure 7.3), the proposed approach automatically selects and tunes an optimal uncertainty estimation schema based on STL-U criteria. As we will discuss in Section 7.4, such uncertainty calibration is an essential step to guarantee the quality of predictive monitoring, in order to better support decision making of CPS. At *runtime* (the flow marked by the blue lines in Figure 7.3), the proposed approach runs as a continuous iterative process to monitor the predicted future states. Considering the predictive monitoring of AQI in a smart city, for example, at time t, the proposed approach first predicts the AQI for the future 3 hours from time t and monitors if the predictions satisfy the requirements; after a period Δt (e.g., 30 minutes), it predicts the AQI for the future 3 hours from $t + \Delta t$ and checks if the new predictions satisfy the requirements. In this way, the proposed approach provides continuous predictive monitoring of future states to support decision making of CPS.

7.3 STL-U Monitor

As described in the previous section, our approach adopts Bayesian RNN models to make sequential predictions about uncertain future states. We propose a *Signal Temporal Logic with Uncertainty* (STL-U) to monitor such uncertain sequential predictions. We introduce STL-U syntax and semantics

in Section 7.3.1, STL-U quantitative semantics in Section 7.3.2 and present methods to compute STL-U confidence guarantees in Section 7.3.3.

7.3.1 STL-U Syntax and Semantics

We formally define a new type of signals called *flowpipes*[2] to represent uncertain sequential predictions generated by Bayesian RNN models. We describe more details about Bayesian RNN models and uncertainty estimation later in Section 7.4.

Definition 7.1 (Flowpipe). *A single-variable flowpipe Ω is defined over a finite discrete time domain $\mathbb{T}$ such that $\Omega[t] = \Phi_t$ at any time $t \in \mathbb{T}$ and Φ_t is a Gaussian distribution $\mathcal{N}(\theta_t, \sigma_t^2)$. Let $\omega : \{\Omega\}^n$ be a (multi-variable) flowpipe signal, where $n = |X|$ is the size of a finite set of (independent) real variables X. Each variable $\mathsf{x} \in X$ has a corresponding flowpipe ω_x whose value at time t follows a Gaussian distribution Φ_t, denoted by $\omega_\mathsf{x}[t] = \Phi_t$.*

Given a confidence level $\varepsilon \in (0,1) \subseteq \mathbb{R}$, a single-variable flowpipe Ω at time t is bounded by a confidence interval $[\Phi_t^-(\varepsilon), \Phi_t^+(\varepsilon)]$ with the lower bound $\Phi_t^-(\varepsilon) = \theta_t - \delta \cdot \frac{\sigma_t}{\sqrt{N}}$ and the upper bound $\Phi_t^+(\varepsilon) = \theta_t + \delta \cdot \frac{\sigma_t}{\sqrt{N}}$, where N is the number of samples that the Gaussian distribution is estimated from, and δ is a function $\delta = F^{-1}(\frac{\varepsilon}{2})$ with F denoting the CDF of the standard normal distribution $\mathcal{N}(0,1)$ [142]. In the special case where the Gaussian distribution's variance is $\sigma_t = 0$, a flowpipe signal becomes a single trace because the lower and upper bounds of the confidence interval coincide (i.e., $\Phi_t^-(\varepsilon) = \Phi_t^+(\varepsilon) = \theta_t$). Given a (multi-variable) trace $\bar{\omega}$ and a flowpipe ω over the same set of real variables X, we say that $\bar{\omega}$ belongs to ω, denoted by $\bar{\omega} \in \omega$, if $\bar{\omega}_\mathsf{x}[t] \in [\Phi_t^-(\varepsilon), \Phi_t^+(\varepsilon)]$ for all $\mathsf{x} \in X$ and $t \in \mathbb{T}$, where $[\Phi_t^-(\varepsilon), \Phi_t^+(\varepsilon)]$ is the confidence interval of flowpipe ω_x under confidence level ε.

Definition 7.2 (STL-U Syntax). *A STL-U formula φ over a flowpipe signal ω is given by*

$$\varphi := \mu_\mathsf{x}(\varepsilon) \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \Box_I \varphi \mid \Diamond_I \varphi \mid \varphi_1 \, \mathcal{U}_I \, \varphi_2$$

where $\mu_\mathsf{x}(\varepsilon)$ is an atomic predicate over variable x with confidence level ε, whose value is determined by $\mu_\mathsf{x}(\varepsilon) \equiv f(x) > 0$ with a continuous function $f(x)$ about flowpipe ω_x under confidence level ε. Temporal operators $\Box_I$, $\Diamond_I$ and $\mathcal{U}_I$ with a time interval $I \subseteq \mathbb{T}$ represent (bounded) "always", "eventually", and "until", respectively.

[2] To be noted, here we use the concept of flowpipes but define it in a new way.

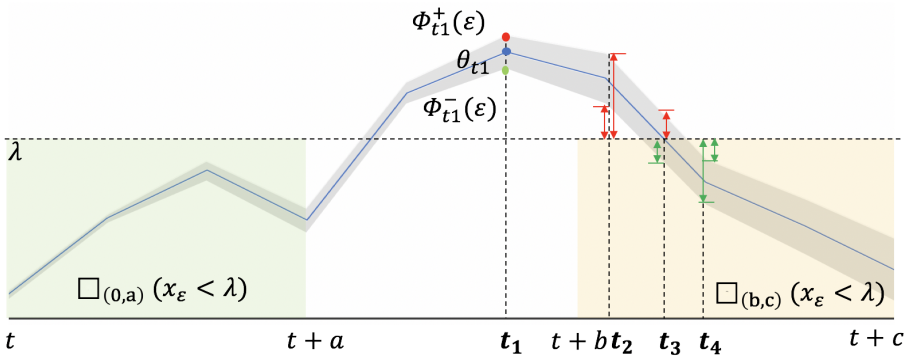


Figure 7.4: An example flowpipe under the confidence level ε.

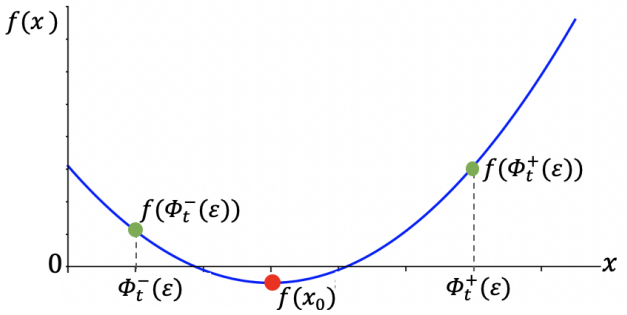


Figure 7.5: An example function $f(x)$.

We define the semantics of a flowpipe signal ω satisfying a STL-U formula φ at time t by two indices: *strong satisfaction*, denoted by $(\omega, t) \vDash_s \varphi$; and *weak satisfaction*, denoted by $(\omega, t) \vDash_w \varphi$.

Definition 7.3. *STL-U strong satisfaction semantics.*

$$
\begin{aligned}
(\omega,t) \vDash_s \mu_x(\varepsilon) &\Leftrightarrow \forall x \in [\Phi^-_t(\varepsilon), \Phi^+_t(\varepsilon)], f(x) > 0 \\
(\omega,t) \vDash_s \neg\varphi &\Leftrightarrow (\omega,t) \nvDash_w \varphi \\
(\omega,t) \vDash_s \varphi_1 \wedge \varphi_2 &\Leftrightarrow (\omega,t) \vDash_s \varphi_1 \text{ and } (\omega,t) \vDash_s \varphi_2 \\
(\omega,t) \vDash_s \square_I \varphi &\Leftrightarrow \forall t' \in (t+I), (\omega,t') \vDash_s \varphi \\
(\omega,t) \vDash_s \Diamond_I \varphi &\Leftrightarrow \exists t' \in (t+I), (\omega,t') \vDash_s \varphi \\
(\omega,t) \vDash_s \varphi_1 \mathcal{U}_I \varphi_2 &\Leftrightarrow \exists t' \in (t+I) \cap \mathbb{T}, (\omega,t') \vDash_s \varphi_2 \text{ and } \forall t'' \in (t,t'), (\omega,t'') \vDash_s \varphi_1
\end{aligned}
$$

Definition 7.4. *STL-U weak satisfaction semantics.*

$$
\begin{aligned}
(\omega,t) \vDash_w \mu_{\mathrm{x}}(\varepsilon) &\Leftrightarrow \exists x \in [\Phi_t^-(\varepsilon), \Phi_t^+(\varepsilon)], f(x) > 0 \\
(\omega,t) \vDash_w \neg\varphi &\Leftrightarrow (\omega,t) \nvDash_s \varphi \\
(\omega,t) \vDash_w \varphi_1 \wedge \varphi_2 &\Leftrightarrow (\omega,t) \vDash_w \varphi_1 \ \textit{and}\ (\omega,t) \vDash_w \varphi_2 \\
(\omega,t) \vDash_w \Box_I \varphi &\Leftrightarrow \forall t' \in (t+I), (\omega,t') \vDash_w \varphi \\
(\omega,t) \vDash_w \Diamond_I \varphi &\Leftrightarrow \exists t' \in (t+I), (\omega,t') \vDash_w \varphi \\
(\omega,t) \vDash_w \varphi_1 \mathcal{U}_I \varphi_2 &\Leftrightarrow \exists t' \in (t+I) \cap \mathbb{T}, (\omega,t') \vDash_w \varphi_2 \ \textit{and}\ \forall t'' \in (t,t'), (\omega,t'') \vDash_w \varphi_1
\end{aligned}
$$

To be noted, the negation of strong satisfaction is equivalent to weak violation, and the negation of weak satisfaction is equivalent to strong violation. Intuitively, strong satisfaction means that all values bounded within the confidence interval of a flowpipe should satisfy the STL-U formula, while weak satisfaction means that there exist some value within the confidence interval of a flowpipe satisfying the STL-U formula. In CPS applications, strong satisfaction relations can be used for monitoring strict requirements (e.g., safety), while weak satisfaction relations can be used for monitoring soft constraints (e.g., energy consumption).

Figure 7.4 shows an example flowpipe signal under the confidence level ε, representing predictions from time t to $t+c$. At a time-point t_1, the flowpipe follows a Gaussian distribution Φ_{t_1} with the mean of θ_{t_1} and is bounded by a confidence interval $[\Phi_{t_1}^-(\varepsilon), \Phi_{t_1}^+(\varepsilon)]$. This flowpipe signal strongly satisfies STL-U formula $\Box_{(0,a)}(x_\varepsilon < \lambda)$ at time t, because the flowpipe signal values bounded within the confidence interval from time t to $t+a$ are all below the threshold λ (see the left green zone in Figure 7.4). Consider another STL-U formula $\Box_{(b,c)}(x_\varepsilon < \lambda)$. As shown in Figure 7.4 (yellow zone in the right), the flowpipe's confidence interval is entirely above the threshold λ at time t_2, partially below λ at time t_3, and entirely below λ at time t_4. Therefore, the flowpipe neither strongly nor weakly satisfies the STL-U formula $\Box_{(b,c)}(x_\varepsilon < \lambda)$ at time t.

Theorem 7.1 (Strength relation theorem)**.** *If a flowpipe ω strongly satisfies a STL-U formula φ at time t, then the weak satisfaction relation also holds. On the other hand, if the flowpipe ω does not*

weakly satisfy a STL-U formula φ at time t, then it would also not strongly satisfy φ. Formally,

$$\begin{aligned}(\omega,t) \vDash_s \varphi \quad &\Rightarrow (\omega,t) \vDash_w \varphi \\ (\omega,t) \nvDash_w \varphi \quad &\Rightarrow (\omega,t) \nvDash_s \varphi\end{aligned}$$

Proof. We just need to prove $(\omega,t) \vDash_s \varphi \Rightarrow (\omega,t) \vDash_w \varphi$ by structural induction below. By contraposition, we have $((\omega,t) \nvDash_w \varphi \Rightarrow (\omega,t) \nvDash_s \varphi) \Leftrightarrow ((\omega,t) \vDash_s \varphi \Rightarrow (\omega,t) \vDash_w \varphi)$.

- Base case μ_{x}: by Definition 7.3, $(\omega,t) \vDash_s \varphi \Leftrightarrow \forall x \in [\Phi_t^-(\varepsilon), \Phi_t^+(\varepsilon)], f(x) > 0$, it indicates that $\exists x \in [\Phi_t^-(\varepsilon), \Phi_t^+(\varepsilon)], f(x) > 0 \Leftrightarrow (\omega,t) \vDash_w \varphi$.

- Inductive case $\neg\varphi$: from inductive hypothesis $(\omega,t) \vDash_s \varphi \Rightarrow (\omega,t) \vDash_w \varphi$ (and consequently $((\omega,t) \nvDash_w \varphi \Rightarrow (\omega,t) \nvDash_s \varphi)$, we need to prove that $(\omega,t) \vDash_s \neg\varphi \Rightarrow (\omega,t) \vDash_w \neg\varphi$.

  We have $(\omega,t) \vDash_s \neg\varphi \Leftrightarrow (\omega,t) \nvDash_w \varphi \Rightarrow (\omega,t) \nvDash_s \varphi \Leftrightarrow (\omega,t) \vDash_w \neg\varphi$.

- Inductive case $\varphi_1 \wedge \varphi_2$: from inductive hypothesis $(\omega,t) \vDash_s \varphi_1 \Rightarrow (\omega,t) \vDash_w \varphi_1$ and $(\omega,t) \vDash_s \varphi_2 \Rightarrow (\omega,t) \vDash_w \varphi_2$, we need to prove that $(\omega,t) \vDash_s \varphi_1 \wedge \varphi_2 \Rightarrow (\omega,t) \vDash_w \varphi_1 \wedge \varphi_2$.

  By Definition 7.3 and Definition 7.4, we have $(\omega,t) \vDash_s \varphi_1 \wedge \varphi_2 \Leftrightarrow (\omega,t) \vDash_s \varphi_1 \wedge (\omega,t) \vDash_s \varphi_2 \Rightarrow (\omega,t) \vDash_w \varphi_1 \wedge (\omega,t) \vDash_w \varphi_2 \Leftrightarrow (\omega,t) \vDash_w \varphi_1 \wedge \varphi_2$.

- Inductive case $\Box_I\varphi$: from inductive hypothesis $(\omega,t) \vDash_s \varphi \Rightarrow (\omega,t) \vDash_w \varphi$, we need to prove that $(\omega,t) \vDash_s \Box_I\varphi \Rightarrow (\omega,t) \vDash_w \Box_I\varphi$.

  By Definition 7.3 and Definition 7.4, we have $(\omega,t) \vDash_s \Box_I\varphi \Leftrightarrow \forall t' \in (t+I), (\omega,t') \vDash_s \varphi$, which indicates that $\forall t' \in (t+I), (\omega,t') \vDash_w \varphi \Leftrightarrow (\omega,t) \vDash_w \Box_I\varphi$.

- Inductive case $\Diamond_I\varphi$: from inductive hypothesis $(\omega,t) \vDash_s \varphi \Rightarrow (\omega,t) \vDash_w \varphi$, we need to prove that $(\omega,t) \vDash_s \Diamond_I\varphi \Rightarrow (\omega,t) \vDash_w \Diamond_I\varphi$.

  By Definition 7.3 and Definition 7.4, we have $(\omega,t) \vDash_s \Diamond_I\varphi \Leftrightarrow \exists t' \in (t+I), (\omega,t') \vDash_s \varphi$, which indicates that $\exists t' \in (t+I), (\omega,t') \vDash_w \varphi \Leftrightarrow (\omega,t) \vDash_w \Diamond_I\varphi$.

- Inductive case $\varphi_1 \mathcal{U}_I \varphi_2$: from inductive hypothesis $(\omega,t) \vDash_s \varphi_1 \Rightarrow (\omega,t) \vDash_w \varphi_1$ and $(\omega,t) \vDash_s \varphi_2 \Rightarrow (\omega,t) \vDash_w \varphi_2$, we prove that $(\omega,t) \vDash_s \varphi_1 \mathcal{U}_I \varphi_2 \Rightarrow (\omega,t) \vDash_w \varphi_1 \mathcal{U}_I \varphi_2$.

By Definition 7.3 and Definition 7.4, we have $(\omega,t) \vDash_s \varphi_1 \mathcal{U}_I \varphi_2 \Leftrightarrow \exists t' \in (t+I) \cap \mathbb{T}, (\omega,t') \vDash_s \varphi_2$ and $\forall t'' \in (t,t'), (\omega,t'') \vDash_s \varphi_1$, which indicates that $\exists t' \in (t+I) \cap \mathbb{T}, (\omega,t') \vDash_w \varphi_2$ and $\forall t'' \in (t,t'), (\omega,t'') \vDash_w \varphi_1$, it is equivalent to $(\omega,t) \vDash_w \varphi_1 \mathcal{U}_I \varphi_2$.

□

Additional properties and examples By applying the rules of the weak/strong semantics for negation we have that the following properties hold:

$$(\omega,t) \vDash_s \neg\neg\varphi \quad \Leftrightarrow \quad (\omega,t) \vDash_s \varphi \qquad (\omega,t) \vDash_w \neg\neg\varphi \quad \Leftrightarrow \quad (\omega,t) \vDash_w \varphi$$

By applying the De Morgan's laws we have the following:

$$\begin{aligned}
(\omega,t) \nvDash_w \neg\varphi_1 \wedge (\omega,t) \nvDash_w \neg\varphi_2 \quad &\Leftrightarrow \quad \neg((\omega,t) \vDash_w (\neg\varphi_1 \vee \neg\varphi_2)) \quad \Leftrightarrow \quad (\omega,t) \nvDash_w \neg(\varphi_1 \wedge \varphi_2) \\
(\omega,t) \nvDash_s \neg\varphi_1 \wedge (\omega,t) \nvDash_s \neg\varphi_2 \quad &\Leftrightarrow \quad \neg((\omega,t) \vDash_s (\neg\varphi_1 \vee \neg\varphi_2)) \quad \Leftrightarrow \quad (\omega,t) \nvDash_s \neg(\varphi_1 \wedge \varphi_2)
\end{aligned}$$

Furthermore, to clarify the duality of the two semantics, we can interpret the weak semantics using the interval intersection and the strong semantics using the interval inclusion. For example, let us define as basic propositions: μ_x^f s.t. $f(x) = x$ and μ_x^g s.t. $g(x) = -x$. Then we have:

$$\begin{aligned}
(\omega,t) \vDash_w \mu_x^f(\epsilon) \quad &\Leftrightarrow \quad \exists x \in [\Phi_t^-(\varepsilon), \Phi_t^+(\varepsilon)], x > 0 \quad \Leftrightarrow \quad [\Phi_t^-(\varepsilon), \Phi_t^+(\varepsilon)] \cap (0, +\infty) \neq \varnothing \\
(\omega,t) \vDash_w \mu_x^g(\epsilon) \quad &\Leftrightarrow \quad \exists x \in [\Phi_t^-(\varepsilon), \Phi_t^+(\varepsilon)], x < 0 \quad \Leftrightarrow \quad [\Phi_t^-(\varepsilon), \Phi_t^+(\varepsilon)] \cap (-\infty, 0) \neq \varnothing \\
(\omega,t) \vDash_s \mu_x^f(\epsilon) \quad &\Leftrightarrow \quad \forall x \in [\Phi_t^-(\varepsilon), \Phi_t^+(\varepsilon)], x > 0 \quad \Leftrightarrow \quad [\Phi_t^-(\varepsilon), \Phi_t^+(\varepsilon)] \subset (0, +\infty) \\
(\omega,t) \vDash_s \mu_x^g(\epsilon) \quad &\Leftrightarrow \quad \forall x \in [\Phi_t^-(\varepsilon), \Phi_t^+(\varepsilon)], x < 0 \quad \Leftrightarrow \quad [\Phi_t^-(\varepsilon), \Phi_t^+(\varepsilon)] \subset (+\infty, 0)
\end{aligned}$$

$$(\omega,t) \vDash_w \neg\mu_x^f(\epsilon) \Leftrightarrow \underbrace{\exists x \in [\Phi_t^-(\varepsilon), \Phi_t^+(\varepsilon)], x \leq 0}_{[\Phi_t^-(\varepsilon), \Phi_t^+(\varepsilon)] \cap (-\infty, 0] \neq \varnothing} \Leftrightarrow \underbrace{\neg(\forall x \in [\Phi_t^-(\varepsilon), \Phi_t^+(\varepsilon)], x > 0)}_{[\Phi_t^-(\varepsilon), \Phi_t^+(\varepsilon)] \not\subset (0, +\infty)} \Leftrightarrow (\omega,t) \nvDash_s \mu_x^f(\epsilon)$$

Following the definition of negation for the weak semantics we have that:

$$(\omega,t) \vDash_w (\mu_x^f(\epsilon) \wedge \mu_x^g(\epsilon)) \quad \Leftrightarrow \quad (\omega,t) \nvDash_s \neg(\mu_x^f(\epsilon) \wedge \mu_x^g(\epsilon))$$

We can show the equivalence consistency using the interval intersection/inclusion interpretation:

$$(\omega,t) \vDash_w (\mu_x^f(\epsilon) \wedge \mu_x^g(\epsilon)) \Leftrightarrow (\omega,t) \vDash_w \mu_x^f(\epsilon) \wedge (\omega,t) \vDash_w \mu_x^g(\epsilon)$$

$$\underbrace{(\omega,t) \vDash_w \mu_x^f(\epsilon)}_{[\Phi_t^-(\varepsilon),\Phi_t^+(\varepsilon)]\cap(0,+\infty)\neq\varnothing} \wedge \underbrace{(\omega,t) \vDash_w \mu_x^g(\epsilon)}_{[\Phi_t^-(\varepsilon),\Phi_t^+(\varepsilon)]\cap(-\infty,0)\neq\varnothing} \Leftrightarrow \underbrace{(\omega,t) \nvDash_s \neg\mu_x^f(\epsilon)}_{[\Phi_t^-(\varepsilon),\Phi_t^+(\varepsilon)]\not\subset(-\infty,0]} \wedge \underbrace{(\omega,t) \nvDash_s \neg\mu_x^g(\epsilon)}_{[\Phi_t^-(\varepsilon),\Phi_t^+(\varepsilon)]\not\subset[0,+\infty)}$$

By applying the De Morgan's laws shown before:

$$(\omega,t) \nvDash_s \neg\mu_x^f(\epsilon) \wedge (\omega,t) \nvDash_s \neg\mu_x^g(\epsilon) \Leftrightarrow (\omega,t) \nvDash_s \neg(\mu_x^f(\epsilon) \wedge \mu_x^g(\epsilon))$$

It is challenging to monitor STL-U strong and weak satisfactions for a flowpipe that contains an infinite set of sequences. Take the atomic predicate $\mu_{\mathsf{x}}(\varepsilon)$ as an example. Based on Definition 7.3, a flowpipe strongly satisfies $\mu_{\mathsf{x}}(\varepsilon)$ iff $f(x) > 0$ for all $x \in [\Phi_t^-(\varepsilon), \Phi_t^+(\varepsilon)]$. It is computationally expensive if not infeasible to exhaustively search through the entire confidence interval. In addition, it does not suffice to only check the lower and upper bounds of the confidence interval when $f(x)$ is a non-monotonic function. Figure 7.5 shows an example where $f(\Phi_t^-(\varepsilon)) > 0$ and $f(\Phi_t^+(\varepsilon)) > 0$, but there is a $x_0 \in [\Phi_t^-(\varepsilon), \Phi_t^+(\varepsilon)]$ with $f(x_0) < 0$. We tackle this challenge by computing the minimal value f_{min} of $f(x)$ for $x \in [\Phi_t^-(\varepsilon), \Phi_t^+(\varepsilon)]$ (e.g., via minimization algorithms in [143]). If $f_{\mathsf{min}} > 0$, which implies that $f(x) > 0$ for all $x \in [\Phi_t^-(\varepsilon), \Phi_t^+(\varepsilon)]$, then the flowpipe strongly satisfies $\mu_{\mathsf{x}}(\varepsilon)$. Based on Definition 7.4, a flowpipe weakly satisfies $\mu_{\mathsf{x}}(\varepsilon)$ iff there exist some $x \in [\Phi_t^-(\varepsilon), \Phi_t^+(\varepsilon)]$ such that $f(x) > 0$. For monitoring weak satisfaction, we compute the maximal value f_{max} of $f(x)$ for any $x \in [\Phi_t^-(\varepsilon), \Phi_t^+(\varepsilon)]$ and check if $f_{\mathsf{max}} > 0$. We include pseudo code of monitoring algorithms for STL-U strong and weak satisfactions as Algorithm 7.1 and Algorithm 7.2.

Following Definition 7.1, we use an array of triplets $\langle t, \theta_t, \sigma_t \rangle$, $t \in \mathbb{T}$ to represent a flowpipe in STL-U monitoring algorithms. Given θ, σ, and ε, calculating $\Phi_t^+(\varepsilon)$ and $\Phi_t^-(\varepsilon)$ takes a constant time $O(1)$, which could be further accelerated by caching the intermediate results. The time complexity of calculating the predicate $f(x) > 0$ depends on the complexity of $f(x)$ and the selected minimization algorithms. The time complexity of STL-U monitoring algorithms is similar to STL monitoring algorithms. Thus, STL-U can be used to monitor complex specifications (e.g., with multiple levels of nesting temporal operators) via using Algorithm 1 or Algorithm 2 recursively.

7.3.2 Quantitative Semantics.

Let $I = [I^-, I^+]$ be a real-valued interval with $I^- \leq I^+ \in \mathbb{R}$. We define a set of arithmetic operators over intervals: $-_* I \equiv [-I^+, -I^-]$ for the negation of an interval, $\mathsf{min}_*\{I_1, ..., I_n\} \equiv [\min\{I_1^-, ..., I_n^-\}, \min\{I_1^+, ..., I_n^+\}]$

Algorithm 7.1: STL-U strong satisfaction monitoring algorithm $\mathsf{StrongSat}(\varphi, \omega, t)$

```
Function StrongSat(φ, ω, t):
    begin
        switch φ do
            Case μ_x(ε)
                if minimize(f(x), Φ_t^-(ε), Φ_t^+(ε)) > 0 then
                    return True ;
                else
                    return False ;
                end
            Case ¬φ
                return ¬WeakSat(φ, ω, t);
            Case φ_1 ∧ φ_2
                return StrongSat(φ, ω, t) ∧ StrongSat(φ, ω, t)
            Case □_I φ
                for t' ∈ (t + I) do
                    if ¬StrongSat(φ, ω, t') then
                        return False;
                    end
                end
                return True ;
            Case ◊_I φ
                for t' ∈ (t + I) do
                    if StrongSat(φ, ω, t') then
                        return True;
                    end
                end
                return False ;
            Case φ_1 U_I φ_2
                for t' ∈ (t + I) do
                    if StrongSat(φ_2, ω, t') then
                        for t'' ∈ [t, t'] do
                            if ¬StrongSat(φ_1, ω, t'') then
                                return False ;
                            end
                        end
                        return True ;
                    end
                end
                return False ;
        end
    end
```

and $\mathsf{max}_*\{I_1, ..., I_n\} \equiv [\max\{I_1^-, ..., I_n^-\}, \max\{I_1^+, ..., I_n^+\}]$ for the minimization and maximization over a set of intervals $\{I_1, ..., I_n\}$, respectively.

We define the quantitative semantics of a multidimensional flowpipe signal ω satisfying a STL-U formula φ as a *robustness interval* $\rho(\varphi, \omega, t)$ as follows.

Algorithm 7.2: STL-U weak satisfaction monitoring algorithm $\mathsf{WeakSat}(\varphi, \omega, t)$

```
Function WeakSat(φ, ω, t):
    begin
        switch φ do
            Case μ_x(ε)
                if maximize(f(x), Φ_t^-(ε), Φ_t^+(ε)) > 0 then
                    return True ;
                else
                    return False ;
                end
            Case ¬φ
                return ¬StrongSat(φ, ω, t);
            Case φ_1 ∧ φ_2
                return WeakSat(φ, ω, t) ∧ WeakSat(φ, ω, t)
            Case □_I φ
                for t' ∈ (t + I) do
                    if ¬WeakSat(φ, ω, t') then
                        return False;
                    end
                end
                return True ;
            Case ◊_I φ
                for t' ∈ (t + I) do
                    if WeakSat(φ, ω, t') then
                        return True;
                    end
                end
                return False ;
            Case φ_1 U_I φ_2
                for t' ∈ (t + I) do
                    if WeakSat(φ_2, ω, t') then
                        for t'' ∈ [t, t'] do
                            if ¬WeakSat(φ_1, ω, t'') then
                                return False ;
                            end
                        end
                        return True ;
                    end
                end
                return False ;
        end
    end
```

Definition 7.5 (Robustness Interval)**.**

$$\rho(\mu_x(\varepsilon), \omega, t) = [\mathsf{min} f(x), \mathsf{max} f(x)]$$

$$\rho(\neg\varphi, \omega, t) = -_* \rho(\varphi, \omega, t)$$

$$\rho(\varphi_1 \wedge \varphi_2, \omega, t) = \mathsf{min}_*\{\rho(\varphi_1, \omega, t),\ \rho(\varphi_2, \omega, t)\}$$

$$\rho(\Box_I \varphi, \omega, t) = \underset{t' \in (t, t+I)}{\mathsf{min}_*} \rho(\varphi, \omega, t')$$

$$\rho(\Diamond_I \varphi, \omega, t) = \underset{t' \in (t, t+I)}{\mathsf{max}_*} \rho(\varphi, \omega, t')$$

$$\rho(\varphi_1 \mathcal{U}_I \varphi_2, \omega, t) = \underset{t' \in (t, t+I)}{\mathsf{max}_*} \{\mathsf{min}_*\{\rho(\varphi_2, \omega, t'), \underset{t'' \in (t, t')}{\mathsf{min}_*} \rho(\varphi_1, \omega, t'')\}\}$$

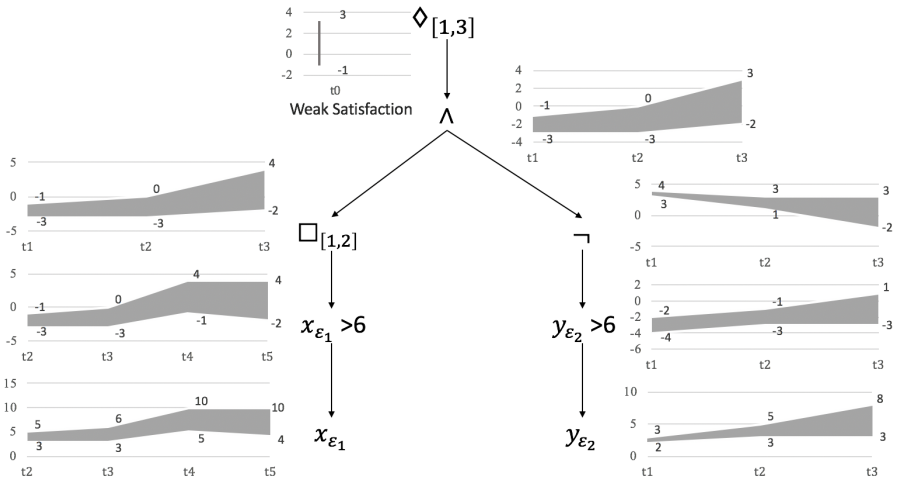


Figure 7.6: Monitoring $\Diamond_{[1,3]}(\Box_{[1,2]}(x_{\varepsilon_1} > 6) \wedge \neg(y_{\varepsilon_2} > 6))$

Intuitively, the quantitative semantics checks the upper and lower bound of the flowpipe. For example, for the always operator $\Box_I$, we calculate the minimum of all discrete values over time interval for both upper and lower bound of the flowpipe. Similarly, for the until operator $\mathcal{U}_I$, to check the robustness interval of φ_1 is always satisfied until φ_2 is satisfied, for each time $t' \in (t, t+I)$ we calculate the robustness interval of φ_2 at time t', and compare it with the minimum robustness interval (i.e. always) of φ_1 when $t'' \in (t, t')$ to get the minimum interval of these two, and finally, we get the maximum interval of all intervals at time $t' \in (t, t+I)$.

Example 7.1. *Figure 7.6 illustrates the processes of calculating the robustness interval for a requirement on multiple flowpipes. First, we parse the requirements and assign the time interval for each condition. Then, from the bottom to the top of the parsing tree as shown in Figure 7.6, We first obtain the flowpipes of x and y under the confidence level ε_1 and ε_2, respectively. For example, the confidence level of x at time t_2 is [3,5]. Next, the robustness interval is calculated for each operation. To check the condition $x_{\varepsilon_1} > 6$, we calculate the robustness interval for each time, for example, it is [-3,-1] at t_2. In the end, we obtain the robustness interval as $\rho = [-1, 3]$, which is a weak satisfaction.*

Theorem 7.2 (Soundness)**.** *Let φ be an STL-U formula and ω be a multidimensional flowpipe signal. Let $\rho(\varphi, \omega, t) = [\rho^-, \rho^+]$ be the robustness interval of ω satisfying φ at time t.*

$$\rho^-(\varphi,\omega,t) > 0 \Rightarrow (\omega,t) \vDash_s \varphi \qquad \rho^-(\varphi,\omega,t) \leq 0 \Rightarrow (\omega,t) \nvDash_s \varphi$$

$$\rho^+(\varphi,\omega,t) > 0 \Rightarrow (\omega,t) \vDash_w \varphi \qquad \rho^+(\varphi,\omega,t) \leq 0 \Rightarrow (\omega,t) \nvDash_w \varphi$$

Proof. We prove the first property $\rho^-(\varphi,\omega,t) > 0 \Rightarrow (\omega,t) \vDash_s \varphi$ by induction:

First we show the soundness property hold for the predicate $\varphi := \mu_x(\varepsilon)$. In this case, we have $\rho^-(\varphi,\omega,t) = f(\Phi_t^+(\varepsilon))$. Therefore, if $\rho^-(\varphi,\omega,t) > 0$ we have $f(\Phi_t^+(\varepsilon)) > 0$, that is, $(\omega,t) \vDash_s \varphi$.

Case $\varphi = \neg\varphi'$: We have $\rho^-(\varphi,\omega,t) = -\rho^+(\varphi',\omega,t) > 0$. Therefore we have $\rho^+(\varphi',\omega,t) < 0$, that is, $(\omega,t) \nvDash_w \varphi'$, which is equivalent to $(\omega,t) \vDash_s \varphi$ by definition.

Case $\varphi = \varphi_1 \wedge \varphi_2$: We have $\rho^-(\varphi_1 \wedge \varphi_2,\omega,t) = \mathsf{min}_*\{\rho^-(\varphi_1,\omega,t),\rho^-(\varphi_2,\omega,t)\} > 0$. Therefore, we have $\rho^-(\varphi_1,\omega,t) > 0$ and $\rho^-(\varphi_1,\omega,t) > 0$. Thus, $(\omega,t) \vDash_s \varphi_1$ and $(\omega,t) \vDash_s \varphi_2$. By definition, we have $(\omega,t) \vDash_s \varphi$.

Case $\varphi = \varphi_1 \mathcal{U}_I \varphi_2$: $\rho^- = \underset{t'\in(t,t+I)}{\mathsf{max}_*}\{\mathsf{min}_*\{\rho^-(\varphi_2,\omega,t'), \underset{t''\in(t,t')}{\mathsf{min}_*}\rho^-(\varphi_1,\omega,t'')\}\} > 0$. We have $\exists t' \in (t+I), \mathsf{min}_*\{\rho^-(\varphi_2,\omega,t'), \underset{t''\in(t,t')}{\mathsf{min}_*}\rho^-(\varphi_1,\omega,t'')\} > 0$. Therefore, $\exists t' \in (t+I), \rho^-(\varphi_2,\omega,t') > 0 \wedge \underset{t''\in(t,t')}{\mathsf{min}_*}\rho^-(\varphi_1,\omega,t'') > 0$. Thus, it's equivalent to $\exists t' \in (t+I)\cap\mathbb{T}, (\omega,t') \vDash_s \varphi_2$ and $\forall t'' \in (t,t'), (\omega,t'') \vDash_s \varphi_1$. By definition, we have $(\omega,t) \vDash_s \varphi$.

The rest of the three properties can be proved similarly. □

If ω_ε strongly(weakly) satisfies φ, any other flowpipe ω'_ε of the same length whose point wise distance (both $\hbar^-$ and $\hbar^+$) from ω_ε is smaller than ρ also strongly(weakly) satisfies φ. We first define the interval distance of two flowpipes below.

Definition 7.6 (Interval Distance)**.** *For any two flowpipes ω_ε and $\omega'_{\varepsilon'}$, the distance between them is also defined as, $[\hbar^-,\hbar^+]$, where*

$$\hbar^-(\omega_\varepsilon,\omega'_{\varepsilon'}) = \min\{\|\Phi_\varepsilon(t)^- - \Phi_\varepsilon(t)^{-'}\|_\infty, \|\Phi_\varepsilon(t)^+ - \Phi_\varepsilon(t)^{+'}\|_\infty\}$$
$$\hbar^+(\omega_\varepsilon,\omega'_{\varepsilon'}) = \max\{\|\Phi_\varepsilon(t)^- - \Phi_\varepsilon(t)^{-'}\|_\infty, \|\Phi_\varepsilon(t)^+ - \Phi_\varepsilon(t)^{+'}\|_\infty\}$$

Theorem 7.3 (Correctness)**.** *Let φ be an STL-U formula, ω_ε a predicted trace with confidence level ε at time t, and $\omega'_{\varepsilon'}$ another predicted trace with confidence level ε' over the same time domain.*

$$(\omega_\varepsilon, t) \vDash_s \varphi \text{ and } \hbar^-(\omega_\varepsilon, \omega'_{\varepsilon'}) < |\rho^-(\varphi, \omega_\varepsilon, t)| \text{ and } \hbar^+(\omega_\varepsilon, \omega'_{\varepsilon'}) < |\rho^-(\varphi, \omega_\varepsilon, t)| \Rightarrow (\omega'_{\varepsilon'}, t) \vDash_s \varphi$$

$$(\omega_\varepsilon, t) \vDash_w \varphi \text{ and } \hbar^-(\omega_\varepsilon, \omega'_{\varepsilon'}) < |\rho^+(\varphi, \omega_\varepsilon, t)| \text{ and } \hbar^+(\omega_\varepsilon, \omega'_{\varepsilon'}) < |\rho^+(\varphi, \omega_\varepsilon, t)| \Rightarrow (\omega'_{\varepsilon'}, t) \vDash_w \varphi$$

Proof. We prove the first property, and the rest of them can be proved by a similar process. By induction, we have the following cases:

case $\varphi := \mu$: We have $\rho^-(\varphi, \omega'_\varepsilon, t) = \inf(\Phi_\varepsilon(t)') \geq \inf(\Phi_\varepsilon(t)) - \hbar^-(\omega_\varepsilon, \omega'_{\varepsilon'}) > 0$. Therefore, we have $(\omega'_{\varepsilon'}, t) \vDash_s \varphi$.

case $\varphi := \neg\varphi'$: We have $\rho^+(\varphi', \omega_\varepsilon, t) = -\rho^-(\varphi, \omega_\varepsilon, t) < -\hbar^-(\omega_\varepsilon, \omega'_{\varepsilon'})$. Similarly, $\rho^+(\varphi', \omega_\varepsilon, t) < -\hbar^+(\omega_\varepsilon, \omega'_{\varepsilon'})$. Therefore, we have $(\omega'_\varepsilon, t) \nvDash_w \varphi'$, which means $(\omega'_\varepsilon, t) \vDash_s \varphi$.

case $\varphi := \varphi_1 \wedge \varphi_2$: Following the definition, we have $(\omega_\varepsilon, t) \vDash_s \varphi_1$ and $(\omega_\varepsilon, t) \vDash_s \varphi_2$. We also have $\rho^-(\varphi, \omega_\varepsilon, t) = \mathsf{min}_*\{\rho^-(\varphi_1, \omega_\varepsilon, t), \rho^-(\varphi_2, \omega_\varepsilon, t)\} > \hbar^-(\omega_\varepsilon, \omega'_{\varepsilon'})$. Therefore, we have $\rho^-(\varphi_1, \omega_\varepsilon, t) > \hbar^-(\omega_\varepsilon, \omega'_{\varepsilon'})$ and $\rho^-(\varphi_2, \omega_\varepsilon, t) > \hbar^-(\omega_\varepsilon, \omega'_{\varepsilon'})$. Similarly, we have $\rho^-(\varphi_1, \omega_\varepsilon, t) > \hbar^+(\omega_\varepsilon, \omega'_{\varepsilon'})$ and $\rho^-(\varphi_2, \omega_\varepsilon, t) > \hbar^+(\omega_\varepsilon, \omega'_{\varepsilon'})$. Therefore, by induction we have $(\omega_{\varepsilon'}, t) \vDash_s \varphi_1$ and $(\omega_{\varepsilon'}, t) \vDash_s \varphi_2$. Finally, we have $(\omega_{\varepsilon'}, t) \vDash_s \varphi$

□

7.3.3 STL-U Confidence Guarantees

It may not always be possible for users to specify a confidence level for a flowpipe *a priori*. It is therefore useful to query about, under what confidence level, a flowpipe is guaranteed to strongly (weakly) satisfy a STL-U formula. We present methods to compute such confidence guarantees as follows.

Let $\epsilon_s(\varphi, \omega, t)$ and $\epsilon_w(\varphi, \omega, t)$ denote the range of confidence levels that guarantee a flowpipe signal ω strongly and weakly satisfying a STL-U formula φ at time t, respectively. Let ϵ_s^c (*resp.* ϵ_w^c) denotes the complement set of ϵ_s (*resp.* ϵ_w) within the interval $(0, 1)$.

Definition 7.7. *Confidence guarantees for STL-U strong satisfaction.*

$$
\begin{aligned}
\epsilon_s(\mu_{\mathsf{x}}, \omega, t) &= \left(0, \int_{\theta_t-\eta}^{\theta_t+\eta} \Phi_t(x)dx\right), \text{where } \eta = \inf\left\{|x-\theta_t| \;\middle|\; f(x) \le 0\right\} \\
\epsilon_s(\neg\varphi, \omega, t) &= \epsilon_w^c(\varphi, \omega, t) \\
\epsilon_s(\varphi_1 \wedge \varphi_2, \omega, t) &= \epsilon_s(\varphi_1, \omega, t) \cap \epsilon_s(\varphi_2, \omega, t) \\
\epsilon_s(\Box_I \varphi, \omega, t) &= \bigcap_{t' \in (t+I)} \epsilon_s(\varphi, \omega, t') \\
\epsilon_s(\Diamond_I \varphi, \omega, t) &= \bigcup_{t' \in (t+I)} \epsilon_s(\varphi, \omega, t') \\
\epsilon_s(\varphi_1 \mathcal{U}_I \varphi_2, \omega, t) &= \bigcup_{t' \in (t+I)} \left\{ \epsilon_s(\varphi_2, \omega, t') \cap \left(\bigcap_{t'' \in (t,t')} \epsilon_s(\varphi_1, \omega, t'') \right) \right\}
\end{aligned}
$$

Definition 7.8. *Confidence guarantees for STL-U weak satisfaction.*

$$
\begin{aligned}
\epsilon_w(\mu_{\mathsf{x}}, \omega, t) &= \left(\int_{\theta_t-\eta}^{\theta_t+\eta} \Phi_t(x)dx, 1\right), \text{ where } \eta = \inf\left\{|x-\theta_t| \;\middle|\; f(x) > 0\right\} \\
\epsilon_w(\neg\varphi, \omega, t) &= \epsilon_s^c(\varphi, \omega, t) \\
\epsilon_w(\varphi_1 \wedge \varphi_2, \omega, t) &= \epsilon_w(\varphi_1, \omega, t) \cap \epsilon_w(\varphi_2, \omega, t) \\
\epsilon_w(\Box_I \varphi, \omega, t) &= \bigcap_{t' \in (t+I)} \epsilon_w(\varphi, \omega, t') \\
\epsilon_w(\Diamond_I \varphi, \omega, t) &= \bigcup_{t' \in (t+I)} \epsilon_w(\varphi, \omega, t') \\
\epsilon_w(\varphi_1 \mathcal{U}_I \varphi_2, \omega, t) &= \bigcup_{t' \in (t+I)} \left\{ \epsilon_w(\varphi_2, \omega, t') \cap \left(\bigcap_{t'' \in (t,t')} \epsilon_w(\varphi_1, \omega, t'') \right) \right\}
\end{aligned}
$$

Theorem 7.4. *Given a flowpipe signal ω and a STL-U formula φ, ω is guaranteed to strongly satisfy φ at time t under a confidence level $\varepsilon \in \epsilon_s(\varphi, \omega, t)$ computed based on Definition 7.7.*

Theorem 7.5. *Given a flowpipe signal ω and a STL-U formula φ, ω is guaranteed to weakly satisfy φ at time t under a confidence level $\varepsilon \in \epsilon_w(\varphi, \omega, t)$ computed based on Definition 7.8.*

Proof of Theorem 7.4 and Theorem 7.5. Mathematically, we want to show that for any ω and t, we have $\forall \varepsilon \in \epsilon_s(\varphi, \omega, t), (\omega, t) \vDash_s \varphi$ under confidence level ε, and for any ω and t, we have $\forall \varepsilon \in \epsilon_w(\varphi, \omega, t), (\omega, t) \vDash_w \varphi$ under confidence level ε . We prove the Theorem 7.4 and Theorem 7.5 inductively. Since in the definition of the strong definition and weak definitions refer to each other, we prove them together. We first study whether Theorem 7.4 is satisfied in each case from the definition. Then, we finish our proof by the axiom of induction. We omit the cases of $\Box$ and $\Diamond$ since they can be derived from the case of $\mathcal{U}_I$.

- When $\varphi = \mu_{\mathsf{x}}$, we show $\forall \omega, \forall t, \forall \varepsilon \in \epsilon_s(\varphi, \omega, t), (\omega, t) \vDash_s \varphi$.

  By Definition 7.7, $\epsilon_s(\varphi, \omega, t) = (0, \int_{\theta_t - \eta}^{\theta_t + \eta} \varphi_t(x)dx)$, where $\eta = \inf\left\{|x - \theta_t| \;\middle|\; f(x) \le 0\right\}$. By the definition of a confidence interval, we have for $\varepsilon \in \epsilon_s(\varphi, \omega, t)$, $\Phi_t^+(\varepsilon) \le \theta_t + \eta$ and $\Phi_t^-(\varepsilon) \ge \theta_t - \eta$, which indicates $[\Phi_t^-(\varepsilon), \Phi_t^+(\varepsilon)] \subseteq [\theta_t - \eta, \theta_t + \eta]$. As $\eta = \inf\left\{|x - \theta_t| \;\middle|\; f(x) \le 0\right\}$, we have $\forall x \in [\Phi_t^-(\varepsilon), \Phi_t^+(\varepsilon)], f(x) > 0$. Therefore, $(\omega, t) \vDash_s \varphi$.

- When $\varphi = \mu_{\mathsf{x}}$, we show $\forall \omega, \forall t, \forall \varepsilon \in \epsilon_w(\varphi, \omega, t), (\omega, t) \vDash_w \varphi$.

  By Definition 7.8, $\epsilon_w(\mu_{\mathsf{x}}, \omega, t) = (\int_{\theta_t - \eta}^{\theta_t + \eta} \varphi_t(x)dx, 1)$, where $\eta = \inf\left\{|x - \theta_t| \;\middle|\; f(x) > 0\right\}$. By the definition of confidence interval, we have for $\varepsilon \in \epsilon_s(\varphi, \omega, t)$, $\Phi_t^+(\varepsilon) \ge \theta_t + \eta$ and $\Phi_t^-(\varepsilon) \le \theta_t - \eta$, which indicates $(\theta_t - \eta, \theta_t + \eta) \subseteq [\Phi_t^-(\varepsilon), \Phi_t^+(\varepsilon)]$. Since $\eta = \inf\left\{|x - \theta_t| \;\middle|\; f(x) > 0\right\}$, we have $\exists x \in [\Phi_t^-(\varepsilon), \Phi_t^+(\varepsilon)], f(x) > 0$. Therefore, $(\omega, t) \vDash_w \varphi$.

- When $\varphi = \neg\varphi_1$, we show $\forall \omega, \forall t, \forall \varepsilon \in \epsilon_w(\varphi_1, \omega, t), (\omega, t) \vDash_w \varphi_1 \Rightarrow \forall \omega, \forall t, \forall \varepsilon \in \epsilon_s(\varphi, \omega, t), (\omega, t) \vDash_s \varphi$.

  By Definition 7.7, we have $\epsilon_s(\neg\varphi, \omega, t) = \epsilon_w^c(\varphi_1, \omega, t)$. Therefore, $\forall \varepsilon \in \epsilon_s(\neg\varphi, \omega, t)$, we have $\varepsilon \in \epsilon_w^c(\varphi_1, \omega, t)$. Then, by the definition of confidence interval we have $(\omega, t) \nvDash_w \varphi_1$ under ε. By the Definition 7.3, $(\omega, t) \vDash_s \varphi$.

- When $\varphi = \neg\varphi_1$, we show $\forall \omega, \forall t, \forall \varepsilon \in \epsilon_s(\varphi_1, \omega, t), (\omega, t) \vDash_s \varphi_1 \Rightarrow \forall \omega, \forall t, \forall \varepsilon \in \epsilon_w(\varphi, \omega, t), (\omega, t) \vDash_s \varphi$.

  By Definition 7.8, we have $\epsilon_w(\neg\varphi, \omega, t) = \epsilon_s^c(\varphi_1, \omega, t)$. Therefore, $\forall \varepsilon \in \epsilon_w(\neg\varphi, \omega, t)$, we have $\varepsilon \in \epsilon_s^c(\varphi_1, \omega, t)$. Then, by the definition of confidence interval we have $(\omega, t) \nvDash_s \varphi_1$ under ε. By the definition 7.4, $(\omega, t) \vDash_w \varphi$.

- $(\varphi = \varphi_1 \wedge \varphi_2) \wedge (\forall\omega, \forall t, \forall\varepsilon \in \epsilon_s(\varphi_1, \omega, t), (\omega, t) \vDash_s \varphi_1) \wedge (\forall\omega, \forall t, \forall\varepsilon \in \epsilon_s(\varphi_2, \omega, t), (\omega, t) \vDash_s \varphi_2) \Rightarrow \forall\omega, \forall t, \forall\varepsilon \in \epsilon_w(\varphi, \omega, t), (\omega, t) \vDash_s \varphi$.

  By Definition 7.7, $\epsilon_s(\neg\varphi, \omega, t) = \epsilon_s(\varphi_1, \omega, t) \cap \epsilon_s(\varphi_2, \omega, t)$. Therefore, $\forall\varepsilon \in \epsilon_s(\varphi_1 \wedge \varphi_2, \omega, t)$, we have $\varepsilon \in \epsilon_s(\varphi_1, \omega, t)$ and $\varepsilon \in \epsilon_s(\varphi_2, \omega, t)$. Then we have $(\omega, t) \vDash_s \varphi_1$ under ε and $(\omega, t) \vDash_s \varphi_2$ under ε, which indicates $(\omega, t) \vDash_w \varphi$ by Definition 7.3.

- When $\varphi = \varphi_1 \wedge \varphi_2$, we show $(\forall\omega, \forall t, \forall\varepsilon \in \epsilon_w(\varphi_1, \omega, t), (\omega, t) \vDash_w \varphi_1) \wedge (\forall\omega, \forall t, \forall\varepsilon \in \epsilon_s(\varphi_2, \omega, t), (\omega, t) \vDash_w \varphi_2) \Rightarrow \forall\omega, \forall t, \forall\varepsilon \in \epsilon_w(\varphi, \omega, t), (\omega, t) \vDash_w \varphi$.

  By Definition 7.8, $\epsilon_w(\neg\varphi, \omega, t) = \epsilon_w(\varphi_1, \omega, t) \cap \epsilon_w(\varphi_2, \omega, t)$. Therefore, $\forall\varepsilon \in \epsilon_w(\varphi_1 \wedge \varphi_2, \omega, t)$, we have $\varepsilon \in \epsilon_w(\varphi_1, \omega, t)$ and $\varepsilon \in \epsilon_w(\varphi_2, \omega, t)$. Then we have $(\omega, t) \vDash_w \varphi_1$ under ε and $(\omega, t) \vDash_w \varphi_2$ under ε, which indicates $(\omega, t) \vDash_w \varphi$ by Definition 7.4.

- When $\varphi = \varphi_1 \mathcal{U}_I \varphi_2$, we show $(\forall\omega, \forall t, \forall\varepsilon \in \epsilon_s(\varphi_1, \omega, t), (\omega, t) \vDash_s \varphi_1) \wedge (\forall\omega, \forall t, \forall\varepsilon \in \epsilon_s(\varphi_2, \omega, t), (\omega, t) \vDash_s \varphi_2) \Rightarrow \forall\omega, \forall t, \forall\varepsilon \in \epsilon_s(\varphi, \omega, t), (\omega, t) \vDash_s \varphi$.

  By Definition 7.7, we have $\epsilon_s(\varphi_1 \mathcal{U}_I \varphi_2, \omega, t) = \bigcup\limits_{t' \in (t+I)} \left\{ \epsilon_s(\varphi_2, \omega, t') \cap \left(\bigcap\limits_{t'' \in (t,t')} \epsilon_s(\varphi_1, \omega, t'') \right) \right\}$. Therefore, for $\varepsilon \in \epsilon_s(\varphi_1 \mathcal{U}_I \varphi_2, \omega, t)$, we have $\exists t' \in (t + I), \varepsilon \in \epsilon_s(\varphi_2, \omega, t') \cap \left(\bigcap\limits_{t'' \in (t,t')} \epsilon_s(\varphi_1, \omega, t'') \right)$. Therefore, for this t' we have $\varepsilon \in \epsilon_s(\varphi_2, \omega, t')$ and $\forall t'' \in (t, t'), \varepsilon \in \epsilon_s(\varphi_1, \omega, t''))$. By the inductive assumption, we have $(\omega, t') \vDash_s \varphi_2$ and $\forall t'' \in (t, t'), (\omega, t'') \vDash_s \varphi_1$. Finally, by definition Definition 7.3, we have $(\omega, t) \vDash_s \varphi_1 \mathcal{U}_I \varphi_2$.

- When $\varphi = \varphi_1 \mathcal{U}_I \varphi_2$, we show $(\forall\omega, \forall t, \forall\varepsilon \in \epsilon_w(\varphi_1, \omega, t), (\omega, t) \vDash_w \varphi_1) \wedge (\forall\omega, \forall t, \forall\varepsilon \in \epsilon_w(\varphi_2, \omega, t), (\omega, t) \vDash_w \varphi_2) \Rightarrow \forall\omega, \forall t, \forall\varepsilon \in \epsilon_w(\varphi, \omega, t), (\omega, t) \vDash_w \varphi$.

  By Definition 7.8, we have $\epsilon_w(\varphi_1 \mathcal{U}_I \varphi_2, \omega, t) = \bigcup\limits_{t' \in (t+I)} \left\{ \epsilon_w(\varphi_2, \omega, t') \cap \left(\bigcap\limits_{t'' \in (t,t')} \epsilon_w(\varphi_1, \omega, t'') \right) \right\}$. Therefore, for $\varepsilon \in \epsilon_w(\varphi_1 \mathcal{U}_I \varphi_2, \omega, t)$, we have $\exists t' \in (t + I), \varepsilon \in \epsilon_w(\varphi_2, \omega, t') \cap \left(\bigcap\limits_{t'' \in (t,t')} \epsilon_w(\varphi_1, \omega, t'') \right)$. Therefore, for this t' we have $\varepsilon \in \epsilon_w(\varphi_2, \omega, t')$ and $\forall t'' \in (t, t'), \varepsilon \in \epsilon_w(\varphi_1, \omega, t''))$. By the inductive assumption, we have $(\omega, t') \vDash_w \varphi_2$ and $\forall t'' \in (t, t'), (\omega, t'') \vDash_w \varphi_1$. Finally, by Definition 7.4, we have $(\omega, t) \vDash_w \varphi_1 \mathcal{U}_I \varphi_2$.

□

In the following, we explain the intuition behind our methods. Figure 7.7(a) plots the normal density curve of a Gaussian distribution Φ_t with the mean θ_t. A confidence level ε represents the probability that the corresponding confidence interval $[\Phi_t^-(\varepsilon), \Phi_t^+(\varepsilon)]$ contains a target value, calculated as the

percentage of the area of the normal density curve. When ε approaches 0, the confidence interval shrinks to a single point θ_t; and when ε approaches 1, the confidence interval expands to $(-\infty, \infty)$. In general, the larger the value of ε, the wider the confidence interval range. For example, Figure 7.7(a) shows confidence intervals for two confidence levels ε_1, ε_2 and $\varepsilon_1 < \varepsilon_2$.

To compute the range of confidence levels that guarantee the strong satisfaction of a STL-U formula φ, we first determine the smallest distance η between the mean θ_t and the set of x values that violate φ. Any x value within the interval $(\theta_t - \eta, \theta_t + \eta)$ should satisfy φ. Thus, we compute the integral $\int_{\theta_t-\eta}^{\theta_t+\eta} \Phi_t(x)dx$ as the upper bound of confidence level guarantee for strong satisfaction, denoted by ϵ_s^+. Under any confidence level $\varepsilon \in (0, \epsilon_s^+)$, the flowpipe is guaranteed to strongly satisfy the STL-U formula φ. In the special case when the mean value θ_t violates φ, we have $\eta = 0$ and $\epsilon_s^+ = 0$; thus, there does not exist a feasible value of ε, under which the flowpipe strongly satisfies φ. Consider a STL-U formula $\varphi_1 : x < \lambda_1$. As shown in Figure 7.7(a), the flowpipe under ε_1 strongly satisfies φ_1 because $\varepsilon_1 \in (0, \epsilon_s^+)$, while the flowpipe under ε_2 does not strongly satisfy φ_1 because $\varepsilon_2 > \epsilon_s^+$.

To compute the range of confidence levels that guarantee the weak satisfaction of a STL-U formula φ, we find the smallest distance η between the mean θ_t and the set of x values that satisfy φ. We compute the integral $\int_{\theta_t-\eta}^{\theta_t+\eta} \Phi_t(x)dx$ as the lower bound of confidence level guarantee for weak satisfaction, denoted by ϵ_w^-. Under any confidence level $\varepsilon \in (\epsilon_w^-, 1)$, the flowpipe is guaranteed to weakly satisfy the STL-U formula φ. When there does not exist any x value satisfying φ, we have $\eta = \infty$ and $\epsilon_w^- = 1$; thus, there does not exist a feasible value of ε, under which the flowpipe weakly satisfies φ. Figure 7.7(b) shows the same Gaussian distribution Φ_t as in Figure 7.7(a). For the STL-U formula $\varphi_2 : x < \lambda_2$, the flowpipe under ε_2 weakly satisfy φ_2 because $\varepsilon_2 \in (\epsilon_w^-, 1)$; however, the flowpipe under ε_1 does not weakly satisfy φ_2, because $\varepsilon_1 < \epsilon_w^-$.

We include pseudo code of algorithms for computing confidence guarantees for STL-U strong and weak satisfaction as Algorithm 7.3 and Algorithm 7.4. Here we use an example to describe the

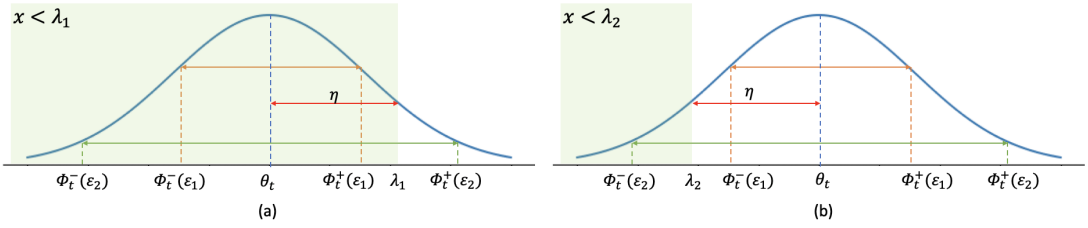


Figure 7.7: Computing confidence guarantees for STL-U formulas $x < \lambda_1$ and $x < \lambda_2$.

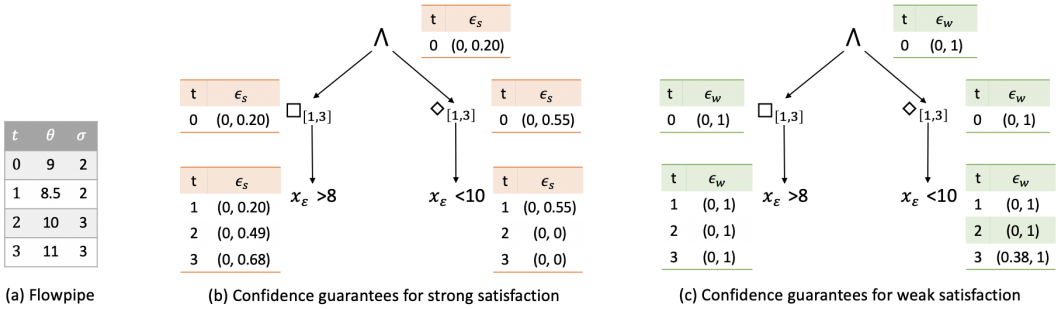


Figure 7.8: Computing confidence guarantees for STL-U formula $\square_{[1,3]}(x_\varepsilon > 8) \wedge \Diamond_{[1,3]}(x_\varepsilon < 10)$.

procedure of recursively computing confidence guarantees via parsing the syntax tree of a STL-U formula. Figure 7.8(a) shows an example flowpipe with values of mean θ and variance σ in each time step t. Consider a STL-U formula $\square_{[1,3]}(x_\varepsilon > 8) \wedge \Diamond_{[1,3]}(x_\varepsilon < 10)$. Figure 7.8(b) illustrates how to iterate through the formula's syntax tree and compute confidence guarantees for strong satisfaction. First, at the left bottom of the tree, we compute the range of confidence levels that can guarantee the strong satisfaction of $x_\varepsilon > 8$, and obtain the results of $(0, 0.20), (0, 0.49), (0, 0.68)$ for $t \in [1, 3]$. Then, we move up the tree to compute the confidence guarantee for $\square_{[1,3]}(x_\varepsilon > 8)$ by taking the intersection of these three ranges, which yields $(0, 0.20)$. Meanwhile, from the right branch of the tree, we obtain the range of confidence levels that guarantee the strong satisfaction of $x_\varepsilon < 10$, and taking a union of these ranges for $\Diamond_{[1,3]}(x_\varepsilon < 10)$, which yields $(0, 0.55)$. At the top of the syntax tree, we take the intersection of $(0, 0.20)$ and $(0, 0.55)$ for $\wedge$ operation, which yields $(0, 0.20)$ as the final result of confidence guarantees for strongly satisfy the STL-U formula. Figure 7.8(c) shows a similar process of recursively computing confidence guarantees for weak satisfaction of the same STL-U formula.

7.4 Prediction with Logic-Calibrated Uncertainty

Recall from Section 7.1 that deterministic prediction models are not suitable to capture the uncertainty exhibited in CPS. To address this limitation, we adopt Bayesian RNN models in the proposed predictive monitoring approach. We describe how to build Bayesian RNN models for prediction and motivate the need for uncertainty calibration in Section 7.4.1. Then, we define STL-U based criteria for uncertainty calibration in Section 7.4.2.

Algorithm 7.3: Confidence Level of Strong Satisfaction $\mathsf{StrongConfidenceLevel}(\varphi, \omega, t)$

Function $\mathsf{StrongConfidenceLevel}(\varphi, \omega, t)$**:**
- **begin**
  - **switch** φ **do**
    - **Case** $\mu_{\times}$
      - $\eta \leftarrow \inf\left\{|x - \theta_t| \,\middle|\, f(x) \leq 0\right\}$
      - **return** $(0, \int_{\theta_t - \eta}^{\theta_t + \eta} \Phi_t(x)dx)$;
    - **Case** $\neg\varphi$
      - $\epsilon_s \leftarrow \mathsf{WeakConfidenceLevel}(\varphi, \omega, t)^C$
      - **return** ϵ_s;
    - **Case** $\varphi_1 \wedge \varphi_2$
      - **return** $\mathsf{StrongConfidenceLevel}(\varphi_1, \omega, t) \cap \mathsf{StrongConfidenceLevel}(\varphi_2, \omega, t)$
    - **Case** $\Box_I \varphi$
      - $\epsilon_s = \mathsf{StrongConfidenceLevel}(\varphi, \Omega, 0)$
      - **for** $t' \in (t + I)$ **do**
        - $\epsilon_s \leftarrow \epsilon_s \cap \mathsf{StrongConfidenceLevel}(\varphi, \Omega, t')$
      - **end**
      - **return** ϵ_s;
    - **Case** $\Diamond_I \varphi$
      - $\epsilon_s \leftarrow \mathsf{StrongConfidenceLevel}(\varphi, \Omega, 0)$
      - **for** $t' \in (t + I)$ **do**
        - $\epsilon_s \leftarrow \epsilon_s \cup \mathsf{StrongConfidenceLevel}(\varphi, \Omega, t')$
      - **end**
      - **return** ϵ_s;
    - **Case** $\varphi_1 \mathcal{U}_I \varphi_2$
      - $\epsilon_s \leftarrow \varnothing$
      - **for** $t' \in (t + I)$ **do**
        - $\epsilon'_s \leftarrow \mathsf{StrongConfidenceLevel}(\varphi_2, \omega, t')$
        - **for** $t'' \in [t, t']$ **do**
          - $\epsilon'_s \leftarrow \epsilon'_s \cap \mathsf{StrongConfidenceLevel}(\varphi_1, \omega, t'')$
        - **end**
        - $\epsilon_s \leftarrow \epsilon_s \cup \epsilon'_s$
      - **end**
      - **return** ϵ_s;
  - **end**
- **end**

7.4.1 Uncertainty Estimation with Bayesian RNN Models

Stochastic regularization techniques (SRTs) have been popularly used to cast deterministic deep learning models as Bayesian models for uncertainty estimation [76]. Given a well-trained deterministic RNN model with learnable parameters W, we can obtain a Bayesian RNN model with parameters W' via SRTs that transform W to W' by applying a $n \times n$ mask, where n is the number of neurons in each layer. Elements of the mask w are sampled from some probability distribution. The connection from neuron j to neuron i would be dropped if $w_{ij} = 0$, the connection remains the same if $w_{ij} = 1$, and a weight $\beta \in (0, 1)$ would be applied to the connection if $w_{ij} = \beta$. In this work, we consider four commonly used SRTs as illustrated in Figure 7.9. Let p denote the dropout rate.

- *Bernoulli dropout*: Each row of the mask is sampled from a Bernoulli distribution, denoted by $w_{i,*} \sim \mathcal{B}(p)$.
- *Bernoulli dropConnect*: Each element of the mask is sampled independently as $w_{i,j} \sim \mathcal{B}(p)$.

Algorithm 7.4: Confidence Level of Weak Satisfaction $\mathsf{WeakConfidenceLevel}(\varphi, \omega, t)$

Function $\mathsf{WeakConfidenceLevel}(\varphi, \omega, t)$**:**
  begin
    switch φ **do**
      Case $\mu_{\times}$
        $\eta \leftarrow \inf\left\{|x - \theta_t| \,\middle|\, f(x) > 0\right\}$
        return $(\int_{\theta_t-\eta}^{\theta_t+\eta} \Phi_t(x)dx, 1)$;
      Case $\neg\varphi$
        $\epsilon_w \leftarrow \mathsf{StrongConfidenceLevel}(\varphi, \omega, t)^C$
        return ϵ_{w};
      Case $\varphi_1 \wedge \varphi_2$
        return $\mathsf{WeakConfidenceLevel}(\varphi_1, \omega, t) \cap \mathsf{WeakConfidenceLevel}(\varphi_2, \omega, t)$
      Case $\Box_I \varphi$
        $\epsilon_w \leftarrow \mathsf{WeakConfidenceLevel}(\varphi, \Omega, 0)$
        for $t' \in (t + I)$ **do**
          $\epsilon_w \leftarrow \epsilon_w \cap \mathsf{WeakConfidenceLevel}(\varphi, \Omega, t')$
        end
        return ϵ_w;
      Case $\Diamond_I \varphi$
        $\epsilon_w \leftarrow \mathsf{WeakConfidenceLevel}(\varphi, \Omega, 0)$
        for $t' \in (t + I)$ **do**
          $\epsilon_w \leftarrow \epsilon_w \cup \mathsf{WeakConfidenceLevel}(\varphi, \Omega, t')$
        end
        return ϵ_w;
      Case $\varphi_1 \mathcal{U}_I \varphi_2$
        $\epsilon_w \leftarrow \varnothing$
        for $t' \in (t + I)$ **do**
          $\epsilon'_w \leftarrow \mathsf{WeakConfidenceLevel}(\varphi_2, \omega, t')$
          for $t'' \in [t, t']$ **do**
            $\epsilon'_w \leftarrow \epsilon'_w \cap \mathsf{WeakConfidenceLevel}(\varphi_1, \omega, t'')$
          end
          $\epsilon_w \leftarrow \epsilon_w \cup \epsilon'_w$
        end
        return ϵ_w;
    end
  end

- *Gaussian dropout*: Each row of the mask is sampled from a Gaussian distribution, denoted by $w_{i,*} \sim \mathcal{N}(1, (1-p)/p)$.

- *Gaussian dropConnect*: Each element of the mask is sampled independently, denoted by $w_{i,j} \sim \mathcal{N}(1, (1-p)/p)$.

Figure 7.10 shows how to use the obtained Bayesian RNN model to predict future states based on historical states. We apply the Monte Carlo method to repeat the Bayesian RNN prediction for N times, which yield a set of sequential predictions. Thus, we can estimate a Gaussian distribution $\Phi_t \sim \mathcal{N}(\theta_t, \sigma_t^2)$ for each time step t, where the mean θ_t and variance σ_t are computed based on the Monte Carlo samples $\{x_t^{(1)}, \cdots, x_t^{(N)}\}$.

Different uncertainty estimation schemas (i.e., SRTs and dropout rates) can yield different uncertainty estimates for the same model trained with the same data. How to select the best uncertainty estimation schema for an application still remains an open question. Currently, a common practice is

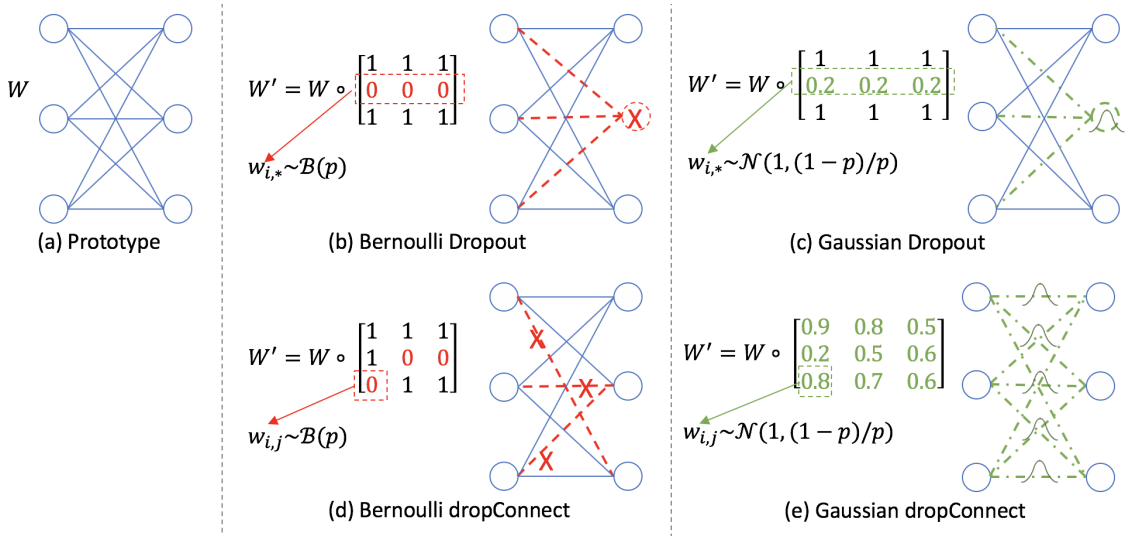


Figure 7.9: Four commonly used SRTs.

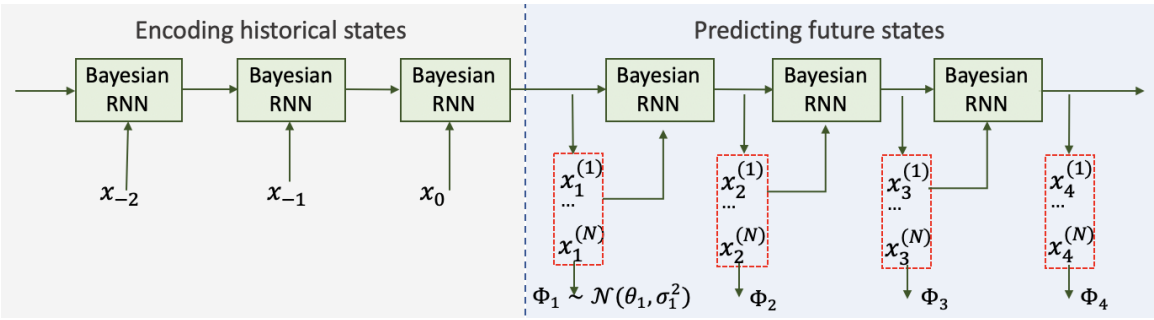


Figure 7.10: Bayesian RNN-based sequential prediction with uncertainty estimation.

to pick a schema empirically, without systematically evaluating how different choices would impact the quality of uncertainty estimates. Furthermore, deep learning methods that seek to optimize the prediction accuracy may overestimate the uncertainty. For example, consider two distributions predicted with uncertainty estimation schemas $\mathcal{M}_1(p_1)$ and $\mathcal{M}_2(p_2)$, as shown in Figure 7.11(a) and (b). $\mathcal{M}_2(p_2)$ is a better schema based on the metric of prediction accuracy, because the target value (red dot) falls within the confidence interval $[\Phi_t^-(\varepsilon), \Phi_t^+(\varepsilon)]$ in Figure 7.11(b), but not in Figure 7.11(a). However, $\mathcal{M}_2(p_2)$ yields a higher level of uncertainty, as indicated by the larger confidence interval range.

In this work, we develop novel criteria that leverage STL-U monitoring results to select uncertainty estimation schemas. Here is an example to explain the intuition behind our approach. Consider two distributions predicted with uncertainty estimation schemas $\mathcal{M}_2(p_2)$ and $\mathcal{M}_3(p_3)$, as shown in Figure 7.11(b) and (c). Both distributions fulfill the accuracy metric, because their confidence

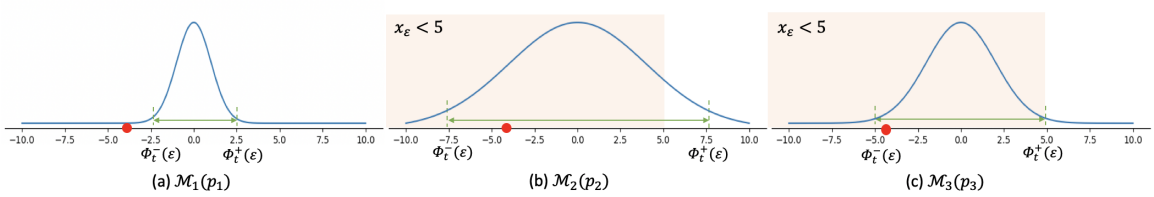


Figure 7.11: Comparison of different uncertainty estimation schemas.

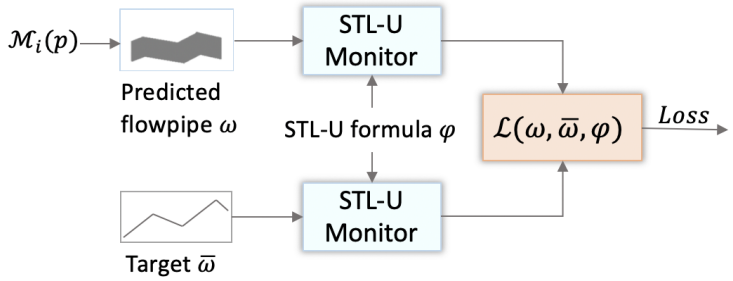


Figure 7.12: STL-U criteria for uncertainty calibration computed as loss functions.

intervals contain the target value (red dot). Suppose that the requirement is to check if a flowpipe strongly satisfies a STL-U formula $x_\varepsilon < 5$. As shown in Figure 7.11(c), the distribution predicted with schema $\mathcal{M}_3(p_3)$ strongly satisfies $x_\varepsilon < 5$, because all values in the confidence interval $[\Phi_t^-(\varepsilon), \Phi_t^+(\varepsilon)]$ are smaller than 5. By contrast, the resulting distribution of schema $\mathcal{M}_2(p_2)$ does not strongly satisfy $x_\varepsilon < 5$, because some values in the confidence interval are greater than 5. Thus, based on STL-U monitoring results, we would select $\mathcal{M}_3(p_3)$ rather than $\mathcal{M}_2(p_2)$ as the uncertainty estimation schema, which also yields a tighter bound of estimated uncertainty. In the following, we formally define STL-U based criteria for selecting uncertainty estimation schemas.

7.4.2 STL-U Criteria for Uncertainty Calibration

As shown in Figure 7.12, given a predicted flowpipe ω and a target trace $\bar{\omega}$, we can calculate the loss based on monitoring results of ω and $\bar{\omega}$ with respect to a STL-U formula φ. We propose two uncertainty calibration criteria as loss functions based on STL-U satisfaction relations and confidence guarantees, denoted by $\mathcal{L}_{\mathsf{sat}}$ and $\mathcal{L}_{\mathsf{cf}}$, respectively.

Criterion based on STL-U satisfaction. We define $\mathcal{L}_{\mathsf{sat}}$ based on the linear combination of three functions: $h_s(\omega, \bar{\omega}, \varphi)$ and $h_w(\omega, \bar{\omega}, \varphi)$ for evaluating if the predicted flowpipe ω and the target trace $\bar{\omega}$ are consistent in terms of strong and weak satisfaction (or violation) of the STL-U formula φ, and $h_b(\omega, \bar{\omega})$ for evaluating the prediction accuracy by checking if the target trace $\bar{\omega}$ belongs to the predicted flowpipe ω. Formally, we define

$$h_s(\omega,\bar{\omega},\varphi) = \mathbb{1}\big((\omega \vDash_s \varphi \wedge \bar{\omega} \vDash \varphi) \vee (\omega \nvDash_s \varphi \wedge \bar{\omega} \nvDash \varphi)\big)$$
$$h_w(\omega,\bar{\omega},\varphi) = \mathbb{1}\big((\omega \vDash_w \varphi \wedge \bar{\omega} \vDash \varphi) \vee (\omega \nvDash_w \varphi \wedge \bar{\omega} \nvDash \varphi)\big)$$
$$h_b(\omega,\bar{\omega}) = \mathbb{1}(\bar{\omega} \in \omega)$$

where $\mathbb{1}(\phi)$ is an indicator function such that $\mathbb{1}(\phi) = 1$ if $\phi = \mathsf{True}$, and $\mathbb{1}(\phi) = 0$ otherwise. The loss function is then given by

$$\mathcal{L}_{\mathsf{sat}}(\omega,\bar{\omega},\varphi) = 1 - (\beta_1 \cdot h_s(\omega,\bar{\omega},\varphi) + \beta_2 \cdot h_w(\omega,\bar{\omega},\varphi) + (1-\beta_1-\beta_2)\cdot h_b(\omega,\bar{\omega}))$$

where $\beta_1, \beta_2 \in (0,1)$ are real-valued coefficients representing the relative importance of strong/weak satisfaction and prediction accuracy in different domains. The goal is to minimize the loss $\mathcal{L}_{\mathsf{sat}}$, for which we need to maximize the linear combination of $h_s(\omega,\bar{\omega},\varphi)$, $h_w(\omega,\bar{\omega},\varphi)$, and $h_b(\omega,\bar{\omega})$. Intuitively, the higher quality of the prediction in terms of the consistency of STL-U monitoring results and the accuracy compared with the target trace, the lower the loss.

Criterion based on STL-U confidence guarantees. Recall from Section 7.3 that, in addition to checking strong/weak satisfaction relations, the STL-U monitor can also compute a range of confidence levels under which the predicted flowpipe is guaranteed to strongly/weakly satisfy a STL-U formula. Based on STL-U confidence guarantees, we define the following loss function:

$$\mathcal{L}_{\mathsf{cf}}(\omega,\bar{\omega},\varphi) = 1 - (\beta_1 \cdot g_s(\omega,\bar{\omega},\varphi) + \beta_2 \cdot g_w(\omega,\bar{\omega},\varphi) + (1-\beta_1-\beta_2)\cdot g_b(\omega,\bar{\omega}))$$

where $\beta_1, \beta_2 \in (0,1)$ are real-valued coefficients similar to those used for $\mathcal{L}_{\mathsf{sat}}$, and $g_s(\omega,\bar{\omega},\varphi)$, $g_w(\omega,\bar{\omega},\varphi)$ and $g_b(\omega,\bar{\omega})$ are functions defined as follows.

$$g_s(\omega, \bar{\omega}, \varphi) = \begin{cases} \epsilon_s^+ & \bar{\omega} \vDash \varphi \\ 1 - \epsilon_s^+ & \bar{\omega} \nvDash \varphi \end{cases}$$

$$g_w(\omega, \bar{\omega}, \varphi) = \begin{cases} 1 - \epsilon_w^- & \bar{\omega} \vDash \varphi \\ \epsilon_w^- & \bar{\omega} \nvDash \varphi \end{cases}$$

$$g_b(\omega, \bar{\omega}) = \inf \{\varepsilon \mid \bar{\omega}_x[t] \in [\Phi_t^-(\varepsilon), \Phi_t^+(\varepsilon)] \text{ for all } x \in X, t \in \mathbb{T}\}$$

where ϵ_s^+ is the upper bound of confidence guarantee for strong satisfaction computed based on Definition 7.7, ϵ_w^- is the lower bound of confidence guarantee for weak satisfaction computed based on Definition 7.8, and $g_b(\omega, \bar{\omega})$ computes the smallest confidence level under which the predicted flowpipe is guaranteed to contain the target trace. The goal is to minimize the loss $\mathcal{L}_{\mathsf{cf}}$, for which we need to maximize the linear combination of $g_s(\omega, \bar{\omega}, \varphi)$, $g_w(\omega, \bar{\omega}, \varphi)$, and $g_b(\omega, \bar{\omega})$. Intuitively, the lower the loss, the higher quality of predictions in terms of confidence guarantees for strong/weak satisfaction and prediction accuracy.

Uncertainty calibration using STL-U criteria. In order to select the best uncertainty estimation schema, we start with a set of candidate schemas $\mathcal{M}_1(p), \mathcal{M}_2(p), ..., \mathcal{M}_n(p)$. For each schema with SRT $\mathcal{M}_i$, we tune the dropout rate parameter p using loss functions $\mathcal{L}_{\mathsf{sat}}$ or $\mathcal{L}_{\mathsf{cf}}$. Given a dataset with multiple target traces, we average the losses over all traces to obtain the optimal dropout rate p^*. We compare the losses of candidate schemas equipped with their corresponding optimal dropout rates, and select the best schema $\mathcal{M}^*(p^*)$ that yields the lowest loss. Such a process of selecting and turning uncertainty estimation schemas based on STL-U criteria is illustrated as part of Figure 7.3.

We evaluate and compare the performance of different STL-U criteria in Section 7.5. Generally speaking, users can choose to use $\mathcal{L}_{\mathsf{sat}}$ or $\mathcal{L}_{\mathsf{cf}}$ depending on their needs and problem domains. For example, we would recommend applications with strict safety requirements (e.g., a fire risk prediction and control service) to adopt $\mathcal{L}_{\mathsf{sat}}$ for checking strong satisfaction relations. By contrast, $\mathcal{L}_{\mathsf{cf}}$ is more flexible and does not require a pre-defined confidence level, which is suitable for applications that do not have a specific confidence level yet try to optimize the uncertainty estimation (e.g., a newly deployed energy control service).

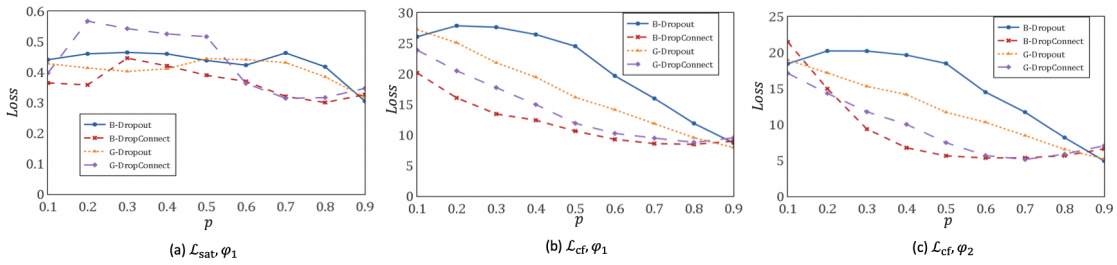


Figure 7.13: Selecting uncertainty estimation schemas using different STL-U criteria.

7.5 Evaluation

We conducted experiments to evaluate the proposed approach. In Section 7.5.1, we compare STL-U criteria for uncertainty calibration with state-of-the-art baselines using real-world CPS datasets. In Section 7.5.2, we demonstrate the performance of our approach on real-time predictive monitoring in a simulated smart city case study. The experiments were run on a machine with 2.2GHz CPU, 32GB memory, and Nvidia GeForce RTX 2080Ti GPU.

7.5.1 Evaluating STL-U Criteria for Uncertainty Calibration

We use two real-world city datasets (i.e., air quality and traffic volume datasets) described in Section 7.1. We split each dataset into 80% data for RNN training, 10% data for STL-U based uncertainty estimation (i.e., tuning Bayesian RNN), and 10% data for testing. We trained the model for 30 epochs.

Comparing different STL-U criteria. Figure 7.13 plots the loss obtained using different STL-U criteria when varying uncertainty estimation schemas (i.e., SRT and dropout rate p) for the air quality dataset. We trained an LSTM as the underlying RNN model. Figure 7.13(a) shows the results of using STL-U criterion $\mathcal{L}_{\text{sat}}$ for $\varphi_1 = \Box_I(x_\varepsilon < \lambda)$, where the schema of Bernoulli DropConnect with $p = 0.8$ yields the lowest loss. Figure 7.13(b) shows the results of using STL-U criterion $\mathcal{L}_{\text{cf}}$ for φ_1, where the schema of Gaussian Dropout with $p = 0.9$ yields the lowest loss. Figure 7.13(c) shows the results of using STL-U criterion $\mathcal{L}_{\text{cf}}$ for $\varphi_2 = \Diamond_I(x_\varepsilon < \lambda)$, where the schema of Bernoulli Dropout with $p = 0.9$ yields the lowest loss. Thus, the optimal uncertainty estimation schema varies based on different STL-U criteria. The experiments demonstrate that the proposed approach is feasible for the automated selection of optimal schemas based on system requirements and user demands (i.e., whether the user is interested in checking requirement satisfaction or computing confidence guarantees).

Table 7.2: Results of comparing STL-U criteria with six baselines.

Criteria	SRT	p	Air Quality HeterLoss	Accuracy	F1-Sat
-	B-Dropout	0.81	183.9	0.67	0.34
-	B-DropConnect	0.53	121.0	0.69	0.22
-	G-Dropout	0.45	152.8	0.76	0.10
-	G-DropConnect	0.58	129.4	0.78	0.12
$\mathcal{L}_{\mathsf{acc}}$	B-DropConnect	0.53	121.0	0.69	0.22
$\mathcal{L}_{\mathsf{ht}}$	G-Dropout	0.50	119.2	0.81	0.65
$\mathcal{L}_{\mathsf{sat}}$	G-DropConnect	0.81	154.1	0.80	**0.81**
$\mathcal{L}_{\mathsf{cf}}$	B-DropConnect	0.73	165.4	0.79	**0.76**
Criteria	**SRT**	p	**Traffic Volume HeterLoss**	**Accuracy**	**F1-Sat**
-	B-Dropout	0.50	0.63	0.79	0.17
-	B-DropConnect	0.74	0.23	0.38	0.51
-	G-Dropout	0.50	0.66	0.79	0.17
-	G-DropConnect	0.54	0.25	0.56	0.44
$\mathcal{L}_{\mathsf{acc}}$	B-DropConnect	0.74	0.23	0.38	0.51
$\mathcal{L}_{\mathsf{ht}}$	G-Dropout	0.50	0.66	0.79	0.17
$\mathcal{L}_{\mathsf{sat}}$	B-DropConnect	0.58	0.24	0.51	**0.67**
$\mathcal{L}_{\mathsf{cf}}$	B-Dropout	0.90	0.3	0.78	**0.68**

Comparing STL-U criteria with baselines. Table 7.2 shows the results of applying uncertainty estimation with STL-U criteria (bottom two rows) and state-of-the-art baselines (top six rows) to the testing data of air quality and traffic volume datasets. We trained an LSTM as the underlying RNN model for each dataset. We consider six *baselines* for comparison. The top four rows of the table are results of using four SRTs with optimal dropout rates p tuned based on the prediction accuracy (i.e., the percentage of target traces covered in the predicted flowpipes). The next two rows are results based on optimizing the uncertainty estimation schema using two commonly used criteria: $\mathcal{L}_{\mathsf{acc}}$ is the loss function concerning the F1-score of prediction accuracy (i.e., if the target trace is covered by the predicted flowpipe), and $\mathcal{L}_{\mathsf{ht}}$ is the loss function approximating the Heteroscedastic aleatoric uncertainty [83]. For the hyperparameters in loss functions, we use $\beta_1 = 0.2$, $\beta_2 = 0.2$ for $\mathcal{L}_{sat}$, and $\beta_1 = 0.3$, $\beta_2 = 0.3$ for $\mathcal{L}_{cf}$.

We compare their performance in terms of three *metrics* shown in columns of the table:

- Heteroscedastic loss: $\mathsf{HeterLoss} = \frac{1}{MT}\sum_{i=1}^{M}\sum_{t=1}^{T}\left(\frac{\|y_t^{(i)}-\theta_t^{(i)}\|^2}{2(\sigma_t^{(i)})^2} + \frac{1}{2}\log 2\sigma_t^{(i)}\right)$, where M represents the total number of instances in the testing data and T represents the length of the predicted sequence;
- Prediction accuracy (RMSE): $\mathsf{Accuracy} = \frac{1}{MT}\sum_{i=1}^{M}\sum_{t=1}^{T}\frac{\|y_t^{(i)}-\theta_t^{(i)}\|^2}{2(\sigma_t^{(i)})^2}$.
- F1-score comparing the STL-U requirement satisfaction for the predicted and target sequences:

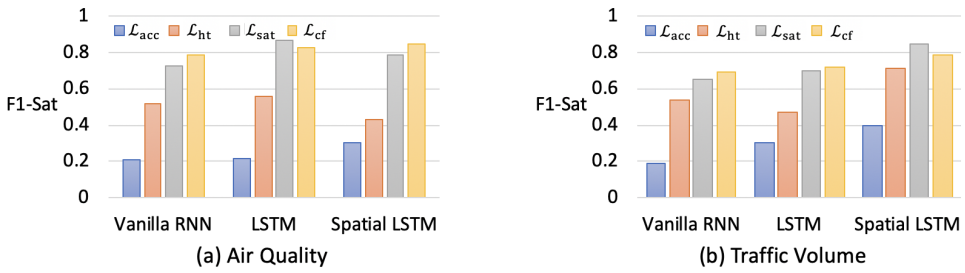


Figure 7.14: Results of comparing different RNN models.

F1-Sat = $\frac{TP}{TP+\frac{1}{2}(FP+FN)}$, where TP, FP, FN represents number of true positives, number of false positives, and number of false negatives, respectively.

The results show that both STL-U criteria yield significant higher F1-scores of requirement satisfaction than all six baselines, which having comparable performance with baselines in terms of Heteroscedastic loss and accuracy. Low F1-scores of requirement satisfaction indicate that flowpipes predicted using baselines can be barely used for monitoring city requirements due to the low quality of estimated uncertainty (i.e., the predicted flowpipes may contain too much noise to obtain meaningful results about requirement violations). Thus, using STL-U criteria to calibrate the uncertainty estimation is an essential step for the predictive monitoring.

Comparing different RNN models. Figure 7.14 compares the F1-score of requirement satisfaction of applying different uncertainty calibration criteria on three types of RNN models: (1) Vanilla RNN, (2) LSTM, and (3) Spatial LSTM [144]. The results show that STL-U criteria $\mathcal{L}_{\mathsf{sat}}$ and $\mathcal{L}_{\mathsf{cf}}$ significantly outperform baseline criteria $\mathcal{L}_{\mathsf{acc}}$ and $\mathcal{L}_{\mathsf{ht}}$ across all three RNN models for both datasets. In addition, both STL-U criteria yield comparable performance across different RNN models. Using $\mathcal{L}_{\mathsf{sat}}$ with an LSTM model and a Spatial LSTM model result in the highest F1-score for the air quality dataset and the traffic volume dataset, respectively. The experiments demonstrate that our proposed approach of uncertainty estimation and calibration is compatible with different underlying RNN models.

7.5.2 Real-time Predictive Monitoring for a Simulated Smart City

We set up a closed-loop simulated smart city based on the *Simulation of Urban MObility* (SUMO) platform [99] using real-world data of New York City [100]. We implemented ten smart services in the simulated smart city, including S1: Traffic Service, S2: Emergency Service, S3: Accident Service, S4: Infrastructure Service, S5: Pedestrian Service, S6: Air Pollution Control Service, S7:

Table 7.3: Results of comparing the impact of STL-U based predictive monitoring with two baselines.

City Performance Metrics	No Monitor	LSTM + STL Monitor	STL-U Predictive Monitor
Number of Violation	-	267	**189**
Air Quality Index	68	57	**43**
Noise (db)	73	49	**48**
Emergency Waiting Time (s)	20	14	**10**
Vehicle Waiting Number	22	18	**15**
Pedestrian Waiting Time (s)	190	148	**121**
Vehicle Waiting Time (s)	112	90	**80**

PM2.5/PM10 Service, S8: Parking Service, S9: Noise Control Service, and S10: Event Service. We built a prototype implementation of the STL-U based predictive monitoring and applied it for the predictive monitoring of 390 requirements concerning different city performance metrics (e.g., AQI, traffic volume, noise) in various locations of the simulated smart city. When a smart service requests an action, we predict future city states under the influence of the requested action and monitor if city requirements would be violated. Based on the real-time predictive monitoring results generated by our approach, the control center can decide if the requested action should be accepted or rejected to prevent any potential requirement violation. For the details of the decision making process, we follow the methods in CityResolver [38], which is a decision making system for conflict detection and resolution in smart cities. As an intuitive example, when a smart navigation service requests an action to direct vehicles to a school area to release traffic congestion, the predictive monitoring approach first predicts the future sequences of noise levels and air pollution levels, and verifies them with STL-U specified city requirements on noise and air pollution in the school areas. If the predicted sequences satisfy the requirements, the requested action will be approved; however, if they violate the requirements, the control center will generate a resolution similar to CityResolver. In our experiment, we run the simulated New York city with STL-U predictive monitoring for 30 simulation days and obtain the results regarding the metrics in Table 7.3. Experimental results show that our approach is efficient in handling a large number of flowpipes and requirements. We did not include the execution time of experiments, because the implementation of our prototype tool is not optimized yet. However, it only takes about 281 seconds to check the satisfaction of 130,000 flowpipes that predict AQI in eight future time units.

Table 7.3 compares STL-U based predictive monitoring's impact on city performance with two *baselines*: (1) running the simulated city without predictive monitoring, and (2) running the simulated city with a basic predictive monitoring component implemented with a deterministic LSTM predictor and a STL monitor. The results are based on 30-day data in the simulated city. First, we observe that our predictive monitor detects fewer requirement violations for the predicted future city states

than the baseline method (2). This is because our approach uses a Bayesian LSTM predictor with calibrated uncertainty, which can generate more accurate predictions about future city states than the deterministic predictor, and thus reducing the number of spurious violations. Furthermore, the results show that our approach has the potential to improve various city performance metrics. For example, compared with the two baselines, our approach reduces the air quality index by 36.8% and 24.6%, and reduces the emergency vehicle waiting time by 50% and 28.6% in the simulated city, respectively.

7.6 Summary

We developed a novel predictive monitoring approach for CPS, which consists of a logic-calibrated Bayesian RNN prediction model that continuously generates sequential predictions of future states, and a novel STL-U monitor that checks if the generated predictions satisfy CPS requirements. Additionally, we proposed novel criteria based on STL-U monitoring results to calibrate uncertainty estimation in Bayesian deep learning for the predictive monitor. The experimental results show that STL-U criteria leads to improved uncertainty estimation in various Bayesian deep learning models, and STL-U based predictive monitor significantly improves performance metrics in a simulated smart city study.

The proposed STL-U monitor is generally applicable for monitoring an infinite set of sequences beyond those generated by Bayesian deep learning. For example, STL-U monitor can also check trajectories of continuous and hybrid systems (e.g., those considered in [73]). In addition, the proposed STL-U criteria for uncertainty calibration can be used in a broad spectrum of deep learning applications. As demonstrated in Section 7.5, STL-U criteria can be used for the automated selection of optimal uncertainty estimation schemas and are compatible with different types of RNN models. Applying STL-U criteria for uncertainty calibration does not require knowledge about the inner working of deep learning models and stochastic regularization techniques. Thus, they are amendable for different deep learning applications.

Chapter 8

Conclusion

Targeting the fundamental challenges in building reliable AI-powered integrated CPS, in this dissertation, we develop rigorous and robust models for reliable i-CPS by integrating formal methods and deep learning. First, we build a decision support system for conflict detection and resolution among integrated IoT services in i-CPS. Additionally, we develop novel formal specification languages and efficient runtime verification techniques to connect verification with the large-scale real-world uncertain environment. Moreover, we develop novel formal methods enhanced deep learning techniques, including increasing the robustness of deep learning prediction by incorporating formal specification and verification into the learning process, and reasoning with uncertainty in deep Bayesian models through logic guided predictive monitoring. Our approaches significantly advances state-of-the-art in many aspects.

Integration of Formal Methods and Deep Learning: This dissertation is a very significant step on integrating formal methods and deep learning towards reliable AI-powered i-CPS. It provides a new perspective towards deep learning challenges. The results show that this approach is effective in both classic and new deep learning models, which means it has the potential to adapt to and guide new deep learning models effectively in the future.

Moreover, it also shows the important role of system properties and requirements in developing deep learning models. There is still a long way to explore and formalize real-world requirements and properties into machine-understandable languages. It also requires the input of human decision-makers who may have no background in formal methods or AI. The specification and monitoring

tool developed in this dissertation is a first step towards cognitive assistant systems for supporting non-expert users.

Meanwhile, the generalizability of this approach is very promising. It is not only applied to different deep learning models and application areas as we have discussed in the previous chapters, it also has the potential to enhance a broad range of cutting-edge research problems through verification of critical system properties and requirements, such as, enhancing algorithmic fairness and equity, robustness certificate, providing a verifiable online learning, etc.

Social and Technological Impact: The theory and models we proposed have potential impacts beyond smart service conflict management to address the fundamental systems-of-systems challenge in multi-stakeholder open environments where each uncertain interaction (e.g., conflicts) between systems can be modeled as a task, and the detection/resolution process of one interaction improves the process of other correlated interactions. Thus, its potential impacts go beyond smart cities including smart rural communities, smart agriculture, and smart health with a generic setting of systems-of-systems with uncertain conflicts. Our social research effort has the potential to understand the roles other technology played with multiple stakeholders involved with issues of diversity, inclusion, and equity.

Bibliography

[1] *IoT and non-IoT connections worldwide 2010-2025*. www.statista.com.

[2] Mansoor Shafi, Andreas F Molisch, Peter J Smith, Thomas Haustein, Peiying Zhu, Prasan De Silva, Fredrik Tufvesson, Anass Benjebbour, and Gerhard Wunder. 5g: A tutorial overview of standards, trials, challenges, deployment, and practice. *IEEE journal on selected areas in communications*, 35(6):1201–1221, 2017.

[3] *IBM Watson IoT Platform*. http://www.ibm.com/internet-of-things/.

[4] *Azure IoT Hub documentation*. https://azure.microsoft.com/en-us/documentation/services/iot-hub/.

[5] *The Internet of Things (IoT) Starts with Intel Inside*. http://www.intel.com/content/www/us/en/internet-of-things/overview.html.

[6] *AWS IoT*. https://aws.amazon.com/iot/.

[7] M Kucharski, A Trujillo, C Dunlop, and B Ahdab. Iso 26262 software compliance: Achieving functional safety in the automotive industry. Technical report, Technical report, 2012.

[8] Thomas Novak, Albert Treytl, and Peter Palensky. Common approach to functional safety and system security in building automation and control systems. In *Emerging Technologies and Factory Automation, 2007. ETFA. IEEE Conference on*, pages 1141–1148. IEEE, 2007.

[9] Sílvia Resendes, Paulo Carreira, and André C Santos. Conflict detection and resolution in home and building automation systems: a literature review. *Journal of Ambient Intelligence and Humanized Computing*, 5(5):699–715, 2014.

[10] Lucia Pallottino, Vincenzo G Scordio, Antonio Bicchi, and Emilio Frazzoli. Decentralized cooperative policy for conflict resolution in multivehicle systems. *IEEE Transactions on Robotics*, 23(6):1170–1183, 2007.

[11] Michael Behrisch, Laura Bieker, Jakob Erdmann, and Daniel Krajzewicz. Sumo–simulation of urban mobility: an overview. In *Proceedings of SIMUL 2011, The Third International Conference on Advances in System Simulation*. ThinkMind, 2011.

[12] Axel Wegener, Michał Piórkowski, Maxim Raya, Horst Hellbrück, Stefan Fischer, and Jean-Pierre Hubaux. Traci: an interface for coupling road traffic and network simulators. In *Proceedings of the 11th communications and networking simulation symposium*, pages 155–163. ACM, 2008.

[13] Sirajum Munir, Mohsin Ahmed, and John Stankovic. Eyephy: Detecting dependencies in cyber-physical system apps due to human-in-the-loop. In *proceedings of the 12th EAI In-*

ternational Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services on 12th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services, pages 170–179. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2015.

[14] Robert L Hester, Alison J Brown, Leland Husband, Radu Iliescu, Drew Pruett, Richard Summers, and Thomas G Coleman. Hummod: a modeling environment for the simulation of integrative human physiology. *Frontiers in physiology*, 2, 2011.

[15] Sarah M. Preum, Md Abu Sayeed Mondol, Meiyi Ma, Hongning Wang, and John A. Stankovic. Preclude: Conflict detection in textual health advice. In *Pervasive Computing and Communications (PerCom), 2017 IEEE International Conference on*. IEEE, 2017.

[16] Sirajum Munir, John Stankovic, et al. Depsys: Dependency aware integration of cyber-physical systems for smart homes. In *Cyber-Physical Systems (ICCPS), 2014 ACM/IEEE International Conference on*, pages 127–138. IEEE, 2014.

[17] Colin Dixon, Ratul Mahajan, Sharad Agarwal, AJ Brush, Bongshin Lee, Stefan Saroiu, and Paramvir Bahl. An operating system for the home. In *Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, pages 337–352, 2012.

[18] Chieh-Jan Mike Liang, Börje F Karlsson, Nicholas D Lane, Feng Zhao, Junbei Zhang, Zheyi Pan, Zhao Li, and Yong Yu. Sift: building an internet of safe things. In *Proceedings of the 14th International Conference on Information Processing in Sensor Networks*, pages 298–309. ACM, 2015.

[19] RM Braga Silva Thais, B Ruiz Linnyer, and AF Loureiro Antonio. How to conciliate conflicting users' interests for different collective, ubiquitous and context-aware applications? In *Local Computer Networks (LCN), 2010 IEEE 35th Conference on*, pages 288–291. IEEE, 2010.

[20] Paulo Carreira, Sílvia Resendes, and André C Santos. Towards automatic conflict detection in home and building automation systems. *Pervasive and Mobile Computing*, 12:37–57, 2014.

[21] Emre Göynügür, Sara Bernardini, Geeth de Mel, Kartik Talamadupula, and Murat Şensoy. Policy conflict resolution in iot via planning. In *Canadian Conference on Artificial Intelligence*, pages 169–175. Springer, 2017.

[22] Geoffrey G Towell, Jude W Shavlik, and Michiel O Noordewier. Refinement of approximate domain theories by knowledge-based neural networks. In *Proceedings of the eighth National conference on Artificial intelligence*, volume 861866. Boston, MA, 1990.

[23] Manoel VM França, Gerson Zaverucha, and Artur S d'Avila Garcez. Fast relational learning using bottom clause propositionalization with artificial neural networks. *Machine learning*, 94(1):81–104, 2014.

[24] Percy Liang, Michael I Jordan, and Dan Klein. Learning from measurements in exponential families. In *Proceedings of the 26th annual international conference on machine learning*, pages 641–648, 2009.

[25] Jun Zhu, Ning Chen, and Eric P Xing. Bayesian inference with posterior regularization and applications to infinite latent svms. *The Journal of Machine Learning Research*, 15(1):1799–1847, 2014.

[26] Jingyi Xu, Zilu Zhang, Tal Friedman, Yitao Liang, and Guy Van den Broeck. A semantic loss function for deep learning with symbolic knowledge. *arXiv preprint arXiv:1711.11157*, 2017.

[27] Russell Stewart and Stefano Ermon. Label-free supervision of neural networks with physics and domain knowledge. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[28] Tim Rocktäschel, Sameer Singh, and Sebastian Riedel. Injecting logical background knowledge into embeddings for relation extraction. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1119–1129, 2015.

[29] Yitao Liang and Guy Van den Broeck. Learning logistic circuits. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4277–4286, 2019.

[30] Ivan Donadello, Luciano Serafini, and Artur D'Avila Garcez. Logic tensor networks for semantic image interpretation. *arXiv preprint arXiv:1705.08968*, 2017.

[31] Yaqi Xie, Ziwei Xu, Mohan S Kankanhalli, Kuldeep S Meel, and Harold Soh. Embedding symbolic knowledge into deep networks. *arXiv preprint arXiv:1909.01161*, 2019.

[32] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *stat*, 1050:9, 2015.

[33] Zhiting Hu, Xuezhe Ma, Zhengzhong Liu, Eduard Hovy, and Eric Xing. Harnessing deep neural networks with logic rules. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, pages 2410–2420, 2016.

[34] Zhiting Hu, Zichao Yang, Xiaodan Liang, Ruslan Salakhutdinov, and Eric P Xing. Toward controlled generation of text. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1587–1596. JMLR. org, 2017.

[35] Bingfeng Chen, Zhifeng Hao, Xiaofeng Cai, Ruichu Cai, Wen Wen, Jian Zhu, and Guangqiang Xie. Embedding logic rules into recurrent neural networks. *IEEE Access*, 7:14938–14946, 2019.

[36] Ezio Bartocci, Jyotirmoy Deshmukh, Alexandre Donzé, Georgios Fainekos, Oded Maler, Dejan Ničković, and Sriram Sankaranarayanan. Specification-based monitoring of cyber-physical systems: a survey on theory, tools and applications. In *Lectures on Runtime Verification*, pages 135–175. Springer, 2018.

[37] Vasumathi Raman, Alexandre Donzé, Mehdi Maasoumy, Richard M Murray, Alberto Sangiovanni-Vincentelli, and Sanjit A Seshia. Model predictive control with signal temporal logic specifications. In *53rd IEEE Conference on Decision and Control*, pages 81–87. IEEE, 2014.

[38] Meiyi Ma, John A Stankovic, and Lu Feng. Cityresolver: a decision support system for conflict resolution in smart cities. In *Proc. of ICCPS 2018*, pages 55–64. IEEE Computer Society / ACM, 2018.

[39] Meiyi Ma, Ezio Bartocci, Eli Lifland, John Stankovic, and Lu Feng. Sastl: Spatial aggregation signal temporal logic for runtime monitoring in smart cities. In *Proc. of ICCPS 2020*, pages 51–62. IEEE, 2020.

[40] Francesca Cairoli, Gianfranco Fenu, Felice Andrea Pellegrino, and Erica Salvato. Model predictive control of glucose concentration based on signal temporal logic specifications. In *2019 6th International Conference on Control, Decision and Information Technologies (CoDIT)*, pages 714–719. IEEE, 2019.

[41] Carolyn L. Talcott. Cyber-physical systems and events. In *Software-Intensive Systems and New Computing Paradigms - Challenges and Visions*, volume 5380 of *LNCS*, pages 101–115. Springer, 2008.

[42] Y. Tan, M. C. Vuran, and S. Goddard. Spatio-temporal event model for cyber-physical systems. In *2009 29th IEEE International Conference on Distributed Computing Systems Workshops*, pages 44–50. IEEE, 2009.

[43] Luués Caires and Luca Cardelli. A spatial logic for concurrency (part i). *Information and Computation*, 186(2):194 – 235, 2003.

[44] Fabrizio Banci Buonamici, Gina Belmonte, Vincenzo Ciancia, Diego Latella, and Mieke Massink. Spatial logics and model checking for medical imaging. *Int. J. Softw. Tools Technol. Transf.*, 22(2):195–217, 2020.

[45] Brandon Bennett, Anthony G. Cohn, Frank Wolter, and Michael Zakharyaschev. Multi-dimensional modal logic as a framework for spatio-temporal reasoning. *Applied Intelligence*, 17(3):239–251, September 2002.

[46] D. Bresolin, P. Sala, D. Della Monica, A. Montanari, and G. Sciavicco. A decidable spatial generalization of metric interval temporal logic. In *2010 17th International Symposium on Temporal Representation and Reasoning*, pages 95–102, 2010.

[47] M. Marx and M. Reynolds. Undecidability of compass logic. *J Logic Computation*, 9(6):897–914, 1999.

[48] I. Haghighi, A. Jones, J. Z. Kong, E. Bartocci, Grosu R., and C. Belta. SpaTeL: A Novel Spatial-Temporal Logic and Its Applications to Networked Systems. In *Proc. of HSCC*, 2015.

[49] Laura Nenzi, Luca Bortolussi, Vincenzo Ciancia, Michele Loreti, and Mieke Massink. Qualitative and quantitative monitoring of spatio-temporal properties. In *Runtime Verification - 6th International Conference, RV 2015*, volume 9333, pages 21–37. Springer, 2015.

[50] O. Maler and D. Nickovic. Monitoring temporal properties of continuous signals. In *Proc. FORMATS*, 2004.

[51] E. Aydin Gol, E. Bartocci, and C. Belta. A formal methods approach to pattern synthesis in reaction diffusion systems. In *Proc. of CDC*, 2014.

[52] Ezio Bartocci, Ebru Aydin Gol, Iman Haghighi, and Calin Belta. A formal methods approach to pattern recognition and synthesis in reaction diffusion networks. *IEEE Trans. Control. Netw. Syst.*, 5(1):308–320, 2018.

[53] Radu Grosu, Scott A. Smolka, Flavio Corradini, Anita Wasilewska, Emilia Entcheva, and Ezio Bartocci. Learning and detecting emergent behavior in networks of cardiac myocytes. *Commun. ACM*, 52(3):97–105, 2009.

[54] Ezio Bartocci, Flavio Corradini, Maria Rita Di Berardini, Emilia Entcheva, Scott A. Smolka, and Radu Grosu. Modeling and simulation of cardiac tissue using hybrid I/O automata. *Theor. Comput. Sci.*, 410(33-34):3149–3165, 2009.

[55] Ezio Bartocci, Luca Bortolussi, Michele Loreti, and Laura Nenzi. Monitoring mobile and spatially distributed cyber-physical systems. In *MEMOCODE 2017: the 15th ACM-IEEE International Conference on Formal Methods and Models for System Design*, pages 146–155. ACM, 2017.

[56] Ezio Bartocci, Luca Bortolussi, Michele Loreti, Laura Nenzi, and Simone Silvetti. Moonlight: A lightweight tool for monitoring spatio-temporal properties. In *Proc. of RV 2020: the 20th International Conference on Runtime Verification*, volume 12399 of *LNCS*, pages 417–428. Springer, 2020.

[57] Luca Bortolussi, Francesca Cairoli, Nicola Paoletti, Scott A. Smolka, and Scott D. Stoller. Neural predictive monitoring. In *Proc. of RV 2019*, volume 11757 of *LNCS*, pages 129–147. Springer, 2019.

[58] Luca Bortolussi, Francesca Cairoli, Nicola Paoletti, Scott A. Smolka, and Scott D. Stoller. Bayesian neural predictive monitoring. In *Proc. of the 2nd Workshop on Artificial Intelligence and Formal Verification, Logic, Automata, and Synthesis*, volume 2785 of *CEUR Workshop Proceedings*, pages 95–100. CEUR-WS.org, 2020.

[59] Reza Babaee, Vijay Ganesh, and Sean Sedwards. Accelerated learning of predictive runtime monitors for rare failure. In *Proc. of RV 2019*, volume 11757 of *LNCS*, pages 111–128. Springer, 2019.

[60] Meiyi Ma, Ji Gao, Lu Feng, and John A Stankovic. Stlnet: Signal temporal logic enforced multivariate recurrent neural networks. In *NeurIPS 2020*, 2020.

[61] Reza Babaee, Arie Gurfinkel, and Sebastian Fischmeister. Predictive run-time verification of discrete-time reachability properties in black-box systems using trace-level abstraction and statistical learning. In *Proc. RV 2018*, volume 11237 of *LNCS*, pages 187–204. Springer, 2018.

[62] Scott D. Stoller, Ezio Bartocci, Justin Seyster, Radu Grosu, Klaus Havelund, Scott A. Smolka, and Erez Zadok. Runtime verification with state estimation. In *Proc. of RV 2011*, volume 7186 of *LNCS*, pages 193–207. Springer, 2012.

[63] Kenan Kalajdzic, Ezio Bartocci, Scott A. Smolka, Scott D. Stoller, and Radu Grosu. Runtime verification with particle filtering. In *Proc. of RV 2013*, volume 8174 of *LNCS*, pages 149–166. Springer, 2013.

[64] Ezio Bartocci, Radu Grosu, Atul Karmarkar, Scott A. Smolka, Scott D. Stoller, Erez Zadok, and Justin Seyster. Adaptive runtime verification. In *Proc. of RV 2012*, volume 7687 of *LNCS*, pages 168–182. Springer, 2012.

[65] Masaki Waga, Étienne André, and Ichiro Hasuo. Model-bounded monitoring of hybrid systems. In *Proc. of ICCPS '21*, pages 21–32. ACM, 2021.

[66] Xin Qin and Jyotirmoy V Deshmukh. Clairvoyant monitoring for signal temporal logic. In *Proc. of FORMATS 2020*, volume 12288 of *LNCS*, pages 178–195. Springer, 2020.

[67] E. Bartocci, J. Deshmukh, A. Donzé, G. Fainekos, O. Maler, D. Nickovic, and S. Sankaranarayanan. Specification-based monitoring of cyber-physical systems: A survey on theory, tools and applications. In *Lectures on Runtime Verification*, volume 10457 of *LNCS*, pages 135–175. Springer, 2018.

[68] Iman Haghighi, Austin Jones, Zhaodan Kong, Ezio Bartocci, Radu Grosu, and Calin Belta. SpaTeL: a novel spatial-temporal logic and its applications to networked systems. In *Proc. of HSCC 2015*, pages 189–198. IEEE, 2015.

[69] Susmit Jha, Vasumathi Raman, Dorsa Sadigh, and Sanjit A. Seshia. Safe autonomy under perception uncertainty using chance-constrained temporal logic. *Journal of Automated Reasoning*, 60(1):43–62, 2018.

[70] Dorsa Sadigh and Ashish Kapoor. Safe control under uncertainty with probabilistic signal temporal logic. In *Proc. of the RSS 2016*, 2016.

[71] Jiwei Li, Pierluigi Nuzzo, Alberto Sangiovanni-Vincentelli, Yugeng Xi, and Dewei Li. Stochastic contracts for cyber-physical system design under probabilistic requirements. In *Proc. of MEMOCODE 2017*, pages 5—-14, 2017.

[72] Panagiotis Kyriakis, Jyotirmoy V. Deshmukh, and Paul Bogdan. Specification mining and robust design under uncertainty: A stochastic temporal logic approach. *ACM Trans. Embed. Comput. Syst.*, 18(5s):96:1–96:21, 2019.

[73] Hendrik Roehm, Jens Oehlerking, Thomas Heinz, and Matthias Althoff. STL model checking of continuous and hybrid systems. In *Proc. of ATVA 2016*, volume 9938 of *LNCS*, pages 412–427. Springer, 2016.

[74] Thao Dang, Tommaso Dreossi, and Carla Piazza. Parameter synthesis through temporal logic specifications. In *Proc. of FM 2015*, volume 9109 of *LNCS*, pages 213–230. Springer, 2015.

[75] Cindy Eisner, Dana Fisman, John Havlicek, Yoad Lustig, Anthony McIsaac, and David Van Campenhout. Reasoning with temporal logic on truncated paths. In *Proc. of CAV 2003*, volume 2725 of *LNCS*, pages 27–39. Springer, 2003.

[76] Yarin Gal. *Uncertainty in deep learning*. PhD thesis, University of Cambridge, 2016.

[77] David JC MacKay. A practical bayesian framework for backpropagation networks. *Neural computation*, 4(3):448–472, 1992.

[78] Geoffrey E Hinton and Drew Van Camp. Keeping the neural networks simple by minimizing the description length of the weights. In *Proc. of COLT 1993*, pages 5–13. ACM, 1993.

[79] Alex Graves. Practical variational inference for neural networks. In *Proc. of NIPS 2011*, pages 2348–2356, 2011.

[80] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *Proc. of ICML 2016*, volume 48, pages 1050–1059. JMLR.org, 2016.

[81] Lingxue Zhu and Nikolay Laptev. Deep and confident prediction for time series at Uber. In *Proc. of ICDM Workshops*, pages 103–110. IEEE, 2017.

[82] Yijun Xiao and William Yang Wang. Quantifying uncertainties in natural language processing tasks. In *Proc. of the AAAI 2019*, volume 33, pages 7322–7329, 2019.

[83] Alex Kendall and Yarin Gal. What uncertainties do we need in bayesian deep learning for computer vision? In *Proc. of NIPS 2017*, pages 5574–5584, 2017.

[84] Charith Perera, Arkady Zaslavsky, Peter Christen, and Dimitrios Georgakopoulos. Sensing as a service model for smart cities supported by internet of things. *Transactions on Emerging Telecommunications Technologies*, 25(1):81–93, 2014.

[85] Luis Sanchez, Luis Muñoz, Jose Antonio Galache, Pablo Sotres, Juan R Santana, Veronica Gutierrez, Rajiv Ramdhany, Alex Gluhak, Srdjan Krco, Evangelos Theodoridis, et al. Smartsantander: Iot experimentation over a smart city testbed. *Computer Networks*, 61:217–238, 2014.

[86] Anu Bask, Karen Spens, Gunnar Stefansson, and Kenth Lumsden. Performance issues of smart transportation management systems. *International Journal of Productivity and Performance Management*, 58(1):55–70, 2008.

[87] Sotiris Zygiaris. Smart city reference model: Assisting planners to conceptualize the building of smart city innovation ecosystems. *Journal of the Knowledge Economy*, 4(2):217–231, 2013.

[88] Jatuporn Chinrungrueng, Udomporn Sunantachaikul, and Satien Triamlumlerd. Smart parking: An application of optical wireless sensor network. In *SAINT Workshops 2007. International Symposium on Applications and the Internet Workshops, 2007.*, pages 66–66. IEEE, 2007.

[89] M. Batty, K. W. Axhausen, F. Giannotti, A. Pozdnoukhov, A. Bazzani, M. Wachowicz, G. Ouzounis, and Y. Portugali. Smart cities of the future. *The European Physical Journal Special Topics*, 214(1):481–518, 2012.

[90] *Safety and Health, U.S. Department of Transportation.* `https://www.transportation.gov/policy/transportation-policy/safety`.

[91] *Air Quality Planning and Standards, U.S. Environmental Protection Agency.* `https://www3.epa.gov/airquality/index.html`.

[92] Alexandre Donzé and Oded Maler. Robust satisfaction of temporal logic over real-valued signals. In *Proceedings of International Conference on Formal Modeling and Analysis of Timed Systems*, pages 92–106, 2010.

[93] Meiyi Ma, Sarah Masud Preum, and John A. Stankovic. Cityguard: A watchdog for safety-aware conflict detection in smart cities. In *Proceedings of the Second International Conference on Internet-of-Things Design and Implementation*, pages 259–270, 2017.

[94] Muhammad Intizar Ali, Feng Gao, and Alessandra Mileo. Citybench: A configurable benchmark to evaluate rsp engines using smart city datasets. In *In proceedings of ISWC 2015 - 14th International Semantic Web Conference*, pages 374–389, Bethlehem, PA, USA, 2015. W3C.

[95] Cisco. The digital value of smart cities, 2017.

[96] New York Times. Ibm takes 'smarter cities' concept to rio de janeiro, 2012.

[97] Cisco. Smart+connected operations center.

[98] Meiyi Ma, John A. Stankovic, and Lu Feng. Runtime monitoring of safety and performance requirements in smart cities. In *1st ACM Workshop on the Internet of Safe Things*, 2017.

[99] *SUMO – Simulation of Urban MObility.* `https://nycopendata.socrata.com/`.

[100] *New York City Open Data.* `https://nycopendata.socrata.com/`.

[101] Gurobi Optimization. `http://www.gurobi.com`.

[102] Tommaso Dreossi, Alexandre Donzé, and Sanjit A Seshia. Compositional falsification of cyber-physical systems with machine learning components. *Journal of Automated Reasoning*, 63(4):1031–1053, 2019.

[103] Yuxuan Liang, Songyu Ke, Junbo Zhang, Xiuwen Yi, and Yu Zheng. Geoman: Multi-level attention networks for geo-sensory time series prediction. In *Proc. of IJCAI 2018*, pages 3428–3434. ijcai.org, 2018.

[104] Yin Zhang, Meikang Qiu, Chun-Wei Tsai, Mohammad Mehedi Hassan, and Atif Alamri. Health-cps: Healthcare cyber-physical system assisted by cloud and big data. *IEEE Systems Journal*, 11(1):88–95, 2015.

[105] Meiyi Ma, Sarah Preum, Mohsin Ahmed, William Tärneberg, Abdeltawab Hendawi, and John Stankovic. Data sets, modeling, and decision making in smart cities: A survey. *ACM Transactions on Cyber-Physical Systems*, 4(2):1–28, 2019.

[106] Alexandre Donzé and Oded Maler. Robust satisfaction of temporal logic over real-valued signals. In *Proc. of FORMATS 2010: the 8th International Conference on Formal Modeling and Analysis of Timed System*, volume 6246 of *LNCS*, pages 92–106. Springer, 2010.

[107] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[108] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[109] Xiaoqing Jin, Alexandre Donzé, Jyotirmoy V Deshmukh, and Sanjit A Seshia. Mining requirements from closed-loop control models. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 34(11):1704–1717, 2015.

[110] Charles E Catlett, Peter H Beckman, Rajesh Sankaran, and Kate Kusiak Galvin. Array of things: a scientific research instrument in the public way: platform design and early lessons learned. In *Proceedings of the 2nd International Workshop on Science of Smart City Operations and Platforms Engineering*, pages 26–33. ACM, 2017.

[111] Zipei Fan, Xuan Song, Tianqi Xia, Renhe Jiang, Ryosuke Shibasaki, and Ritsu Sakuramachi. Online deep ensemble learning for predicting citywide human mobility. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 2(3):105, 2018.

[112] Huichu Zhang, Yu Zheng, and Yong Yu. Detecting urban anomalies using multiple spatio-temporal data sources. *ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 2(1):54, 2018.

[113] Shili Sheng, Erfan Pakdamanian, Kyungtae Han, BaekGyu Kim, Prashant Tiwari, Inki Kim, and Lu Feng. A case study of trust on autonomous driving. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pages 4368–4373. IEEE, 2019.

[114] Iman Haghighi, Austin Jones, Zhaodan Kong, Ezio Bartocci, Radu Gros, and Calin Belta. Spatel: a novel spatial-temporal logic and its applications to networked systems. In *Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control*, pages 189–198. ACM, 2015.

[115] Laura Nenzi, Ezio Bartocci, Luca Bortolussi, Michele Loreti, and Ennio Visconti. Monitoring spatio-temporal properties (invited tutorial). In *Proc. of RV 2020: the 20th International Conference on Runtime Verification*, volume 12399 of *LNCS*, pages 21–46. Springer, 2020.

[116] NYC.gov. Emissions from transportation, nyc environment protection. 2019.

[117] District of Columbia Municipal Regulations and District of Columbia Register. Air quality - motor vehicular pollutants, lead, odors, and nuisance pollutants. 2016.

[118] Steven Matteo and Justin Brannan. A local law to amend the administrative code of the city of new york, in relation to restricting the use of bus lanes by sight-seeing buses. In *Restricting the use of bus lanes by sight-seeing buses*. The New York City Council, 2019.

[119] NYC Environment Protection. Use of heating oil remaining in tanks. The city of New York, 2019.

[120] United States Environmental Protection Agency. Residential energy efficiency. In *Energy Resources for State and Local Governments*. The city of New York, 2019.

[121] San Francisco. San Francisco noise pollution construction. In *San Francisco American Legal Publishing*, 2019.

[122] LA Sec 111.03. Minimum Ambient Noise Level. Official city of los angeles municipal code. 2016.

[123] Hong Kong. Guide to indoor air quality management in hong kong regional offices and public places. In *Guide to Indoor Air Quality Management*, 2019.

[124] NYC.gov. Stopping, standing or parking prohibited in specified places. In *New York Public Law*, 2016.

[125] Beijing Emergency Agency. Pre-hospital medical emergency regulations. 2016.

[126] Beijing Government. Safety management for kindergarten, primary and secondary school. 2016.

[127] Richard E Ladner and Michael J Fischer. Parallel prefix computation. *Journal of the ACM (JACM)*, 27(4):831–838, 1980.

[128] Alexandre Donzé, Thomas Ferrere, and Oded Maler. Efficient robust monitoring for stl. In *Proceedings of International Conference on Computer Aided Verification*, pages 264–279. Springer, 2013.

[129] George S. Lueker. A data structure for orthogonal range queries. In *19th Annual Symposium on Foundations of Computer Science*, pages 28–34. IEEE Computer Society, 1978.

[130] City of Chicago. Crimes of Chicago - one year prior to present. https://data.cityofchicago.org/Public-Safety/Crimes-Map/dfnk-7re6, 2018.

[131] Meiyi Ma, S Masud Preum, W Tarneberg, Moshin Ahmed, Matthew Ruiters, and John Stankovic. Detection of runtime conflicts among services in smart cities. In *2016 IEEE International Conference on Smart Computing (SMARTCOMP)*, pages 1–10. IEEE, 2016.

[132] Stefan Bischof, Athanasios Karapantelakis, Cosmin-Septimiu Nechifor, Amit P Sheth, Alessandra Mileo, and Payam Barnaghi. Semantic modelling of smart city data. 2014.

[133] Eslam Montaser, José-Luis Díez, and Jorge Bondia. Stochastic seasonal models for glucose prediction in the artificial pancreas. *Journal of diabetes science and technology*, 11(6):1124–1131, 2017.

[134] Erfan Pakdamanian, Shili Sheng, Sonia Baee, Seongkook Heo, Sarit Kraus, and Lu Feng. Deeptake: Prediction of driver takeover behavior using multimodal data. In *Proc. CHI '21*, pages 103:1–103:14. ACM, 2021.

[135] Bhavkaran Singh Walia, Qianyi Hu, Jeffrey Chen, Fangyan Chen, Jessica Lee, Nathan Kuo, Palak Narang, Jason Batts, Geoffrey Arnold, and Michael Madaio. A dynamic pipeline for spatio-temporal fire risk prediction. In *Proc. of KDD 2018*, pages 764–773. ACM, 2018.

[136] Ian Zhou, Justin Lipman, Mehran Abolhasan, Negin Shariati, and David W Lamb. Frost monitoring cyber–physical system: A survey on prediction and active protection methods. *IEEE Internet of Things Journal*, 7(7):6514–6527, 2020.

[137] O. Maler and D. Ničković. Monitoring temporal properties of continuous signals. In *Proc. of FORMATS 2004*, volume 3253 of *LNCS*, pages 152–166. Springer, 2004.

[138] Suining He and Kang G Shin. Towards fine-grained flow forecasting: A graph attention approach for bike sharing systems. In *Proc. of WWW 2020*, pages 88–98, 2020.

[139] Lingbo Liu, Zhilin Qiu, Guanbin Li, Qing Wang, Wanli Ouyang, and Liang Lin. Contextualized spatial–temporal network for taxi origin-destination demand prediction. *IEEE Transactions on Intelligent Transportation Systems*, 20(10):3875–3887, 2019.

[140] Yarin Gal, Jiri Hron, and Alex Kendall. Concrete dropout. In *Proc. of NIPS 2017*, pages 3581–3590, 2017.

[141] Steven M Kay. *Fundamentals of statistical signal processing*. Prentice Hall PTR, 1993.

[142] Lothar Sachs. *Applied statistics: a handbook of techniques*. Springer Science & Business Media, 2012.

[143] Richard P Brent. *Algorithms for minimization without derivatives*. Courier Corporation, 2013.

[144] Dejiang Kong and Fei Wu. Hst-lstm: a hierarchical spatial-temporal long-short term memory network for location prediction. In *Proc. of IJCAI 2018*, pages 2341–2347. ijcai.org, 2018.