



Degree Project in Electrical Engineering, specializing in Systems, Control and Robotics

Second cycle, 30 credits

# **Secure-by-Construction Planning and Control of Multi-Agent Systems**

Ensuring Security and Task Satisfaction in Cooperative  
Autonomous Systems

**GEORGIOS MITSOS**



# **Secure-by-Construction Planning and Control of Multi-Agent Systems**

## **Ensuring Security and Task Satisfaction in Cooperative Autonomous Systems**

GEORGIOS MITSOS

Master's Programme, Aerospace Engineering, 120 credits

Date: November 20, 2025

Supervisor: Siyuan Liu

Examiner: Dimos Dimarogonas

School of Electrical Engineering and Computer Science

Swedish title: Secure-by-Construction-planering och -styrning av  
multiagentsystem

Swedish subtitle: Säkerhet och uppgiftsuppfyllelse i kooperativa autonoma  
system



## Abstract

In recent years, ensuring both task satisfaction and security in cyber-physical systems has become increasingly critical, particularly in the context of path planning for systems performing complex Linear Temporal Logic (LTL) tasks. While LTL-based controller synthesis has been widely studied, most existing approaches either address security only post-design or abstract the system as a discrete transition model. This thesis bridges this gap by presenting a unified framework for synthesizing secure-by-construction controllers for both single-agent and multi-agent systems, operating in continuous spaces under affine dynamics.

The framework is initially developed for discrete systems and then extended to handle continuous workspaces using tools from formal methods, control theory, and optimization. Security constraints are integrated directly into the synthesis process, and formal correctness guarantees are provided.

A definition of security is introduced, extending beyond initial-state, current-state,  $K$ -step, and infinite-step opacity. The proposed definition is more general, in the sense that satisfying it implies satisfaction of each of these opacity notions. Security is formalized by ensuring that a passive intruder, capable of observing system outputs but not influencing them, cannot infer sensitive information, such as whether an agent has visited a secret region.

The proposed framework defines three distinct types of multi-agent security: (i) a generalization of single-agent opacity that ensures indistinguishability across agents, (ii) a decoy-based approach, where certain agents conceal the actions of agents executing secret tasks, and (iii) a delay-aware formulation, which accounts for delays in the intruder's observations.

By integrating these security specifications into the control synthesis process, the framework guarantees both satisfaction of LTL tasks and the preservation of security for agents governed by affine dynamics. The correctness and real-world applicability of the proposed algorithms is validated through several single-agent and multi-agent examples.

## Keywords

Secure-by-construction synthesis, Multi-agent systems, Cyber-physical systems, Opacity, Linear Temporal Logic



## Sammanfattning

Under de senaste åren har det blivit allt viktigare att säkerställa både uppgiftsuppfyllelse och säkerhet i cyberfysiska system, särskilt när det gäller vägplanering för system som utför komplexa uppgifter i linjär temporallogik (LTL). Även om LTL-baserad styrenhetsyntes har studerats i stor utsträckning, tar de flesta befintliga metoder antingen upp säkerhet först efter design eller abstraherar systemet som en diskret övergångsmodell. Denna avhandling överbrygger detta gap genom att presentera ett enhetligt ramverk för att syntetisera secure-by-construction-kontroller för både enagentsystem och multiagentsystem, som arbetar i kontinuerliga utrymmen under affin dynamik.

Ramverket utvecklas initialt för diskreta system och utvidgas sedan för att hantera kontinuerliga arbetsytor med hjälp av verktyg från formella metoder, reglerteori och optimering. Säkerhetskrav integreras direkt i syntesprocessen och formella korrekthetsgarantier tillhandahålls.

En definition av säkerhet introduceras, som sträcker sig bortom initialtillstånd, aktuellt tillstånd,  $K$ -steg och oändlig stegsopacitet. Den föreslagna definitionen är mer generell, i den meningen att om den uppfylls innebär det att alla dessa opacitetsbegrepp uppfylls. Säkerhet formaliseras genom att säkerställa att en passiv inkräktare, som kan observera systemets utdata men inte påverka dem, inte kan härleda känslig information, till exempel om en agent har besökt ett hemligt område.

Det föreslagna ramverket definierar tre olika typer av säkerhet för flera agenter: (i) en generalisering av opacitet för enskilda agenter som säkerställer oskiljbarhet mellan agenter, (ii) en decoy-baserad metod, där vissa agenter döljer handlingarna hos agenter som utför hemliga uppgifter, och (iii) en fördröjningsmedveten formulering som tar hänsyn till förseningar i inkräktarens observationer.

Genom att integrera dessa säkerhetsspecifikationer i kontrollsyntesprocessen garanterar ramverket både uppfyllandet av LTL-uppgifter och bevarandet av säkerheten för agenter som styrs av affin dynamik. De föreslagna algoritmernas korrekthet och tillämpbarhet i verkligheten valideras genom flera exempel med en eller flera agenter.

## Nyckelord

Secure-by-construction-syntes, Multiagentsystem, Cyberfysiska system, Opacitet, Linjär temporallogik



## Acknowledgments

I would like to express my gratitude to Siyuan Liu for supervising my thesis. I really appreciate her guidance throughout the semester and our conversations that helped me understand the topic better. It has been a great experience working with her!

Many thanks also to Dimos Dimarogonas for approving the thesis topic and reviewing my work.

And, of course, I cannot thank enough my friends and family back in Greece for always being there for me, and Fratzzy for everything.

Stockholm, November 2025

Georgios Mitsos



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Motivation . . . . .	3
1.2	Problem Statement and Objectives . . . . .	6
1.3	Thesis Contributions . . . . .	7
1.4	Outline of the Thesis . . . . .	8
<b>2</b>	<b>Preliminaries</b>	<b>9</b>
2.1	Weighted Transition Systems . . . . .	10
2.2	Linear Temporal Logic . . . . .	12
2.3	Security . . . . .	15
2.3.1	Weighted Transition System with Security Features . . . . .	15
2.3.2	Language-Based Opacity . . . . .	16
2.3.3	State-Based Opacity . . . . .	17
2.4	Linear Systems . . . . .	19
2.5	Polytopes . . . . .	20
<b>3</b>	<b>Secure-by-Construction Optimal Path Planning for Discrete Systems with LTL Specifications</b>	<b>21</b>
3.1	Optimal Path Planning under LTL Specifications . . . . .	22
3.2	Security in Single-Agent Systems . . . . .	24
3.2.1	Definitions . . . . .	24
3.2.1.1	System Model . . . . .	24
3.2.1.2	Security Property . . . . .	25
3.2.1.3	Construction of Secure WTS . . . . .	25
3.2.2	Security Guarantees . . . . .	27
3.3	Security in Multi-Agent Systems . . . . .	29
3.3.1	Type-A Security . . . . .	29
3.3.1.1	System Model . . . . .	29
3.3.1.2	Security Property . . . . .	31

3.3.1.3	Construction of Secure WTS	31
3.3.2	Type-B Security	34
3.3.2.1	System Model	35
3.3.2.2	Security Property	36
3.3.3	Type-C Security	36
3.3.3.1	System Model	37
3.3.3.2	Security Property	38
3.3.3.3	Construction of Secure WTS	38
3.3.4	Motivating Scenarios for Type-A/B/C Security	42
3.4	Secure Optimal Path Planning under LTL Specifications	45
3.4.1	Complexity Analysis	45
3.5	Motivating Examples	48
3.5.1	Secure Navigation of an Autonomous Robot	48
3.5.2	Secure Navigation of a Multi-Agent Robot Team	51
3.5.2.1	Type-A Security	51
3.5.2.2	Type-B Security	53
3.5.2.3	Type-C Security	55
<b>4</b>	<b>Secure-by-Construction Trajectory Planning for Continuous Linear Systems with LTL Specifications</b>	<b>57</b>
4.1	Single-Agent Systems	58
4.1.1	Problem Setting	58
4.1.1.1	System Model	58
4.1.1.2	Security Property	58
4.1.2	Workspace Partitioning	59
4.1.3	Centroid-Based Transitions	61
4.1.3.1	Construction of the Transition Relation	62
4.1.4	Secure Trajectory Planning under LTL Specifications	66
4.2	Multi-Agent Systems	67
4.2.1	System Model	67
4.2.2	Type-A & Type-B Security	67
4.2.2.1	Security Property	67
4.2.2.2	Construction of Secure WTS	68
4.2.3	Type-C Security	69
4.2.3.1	Security Property	70
4.2.3.2	Construction of Secure WTS	70
4.3	Implementation Notes	75
4.4	Motivating Examples	76
4.4.1	Secure Navigation of an Autonomous Robot	76

4.4.2	Secure Navigation of a Multi-Agent Robot Team . . .	81
<b>5</b>	<b>Conclusions and Future Work</b>	<b>87</b>
5.1	Conclusions . . . . .	88
5.2	Future Work . . . . .	89
	<b>References</b>	<b>91</b>



# List of Figures

3.1	An example WTS with initial states in red and secret states in light blue) . . . . .	50
3.2	The Twin-WTS corresponding to the WTS in Figure 3.1 . . . . .	51
3.3	The Twin-sWTS corresponding to the WTS in Figure 3.1 . . . . .	52
3.4	The Global WTS constructed from two WTSs, both identical to the one in Figure 3.1 . . . . .	53
3.5	Optimal paths with and without Type-B security guarantees . . . . .	55
3.6	Type-C secure optimal paths for all $k \geq 2$ . . . . .	56
4.1	Division of the workspace $P$ into sub-polytopes based on predicates . . . . .	77
4.2	Partition of the workspace $P$ into sub-polytopes based on predicates, initial and secret regions, and observation equivalence classes . . . . .	79
4.3	Partition of the workspace $P'$ into sub-polytopes based on predicates, initial and secret regions, and observation equivalence classes . . . . .	83



# List of Tables

3.1	Comparison of intruder capabilities for Type-A/B/C security . . . . .	42
3.2	Optimal and copy paths with shortest distances for different critical agent assignments and Type-A security . . . . .	54
4.1	Description of the regions in the workspace (Single-agent scenario) . . . . .	77
4.2	Description of the regions in the workspace (Multi-agent scenario) . . . . .	82



# List of Algorithms

1	Optimal Path Planning . . . . .	23
2	Secure Optimal Path Planning . . . . .	46



# **Chapter 1**

## **Introduction**

## Notation

Throughout this thesis, matrices are denoted by bold uppercase letters, and vectors by bold lowercase letters. The symbol  $\mathbf{0}$  denotes a vector or a matrix of appropriate dimension whose entries are all zero.

Moreover, a distinction between continuous and discrete is made in two senses. Continuous and discrete space describe if the system evolves in a continuous state space, e.g., in  $\mathbb{R}^N$ , or transitions through discrete states, e.g., nodes of a graph. Continuous-time and discrete-time dynamics describe if the system evolves according to continuous-time differential equations or discrete-time updates.

## 1.1 Motivation

As autonomous systems increasingly operate in safety-critical and security-critical domains, there is a growing demand for planning and control methods that offer formal correctness guarantees. Applications such as self-driving vehicles, drone swarms, warehouse automation, and intelligent transportation systems require agents to execute complex tasks in dynamic, partially unknown, and potentially adversarial environments. In such settings, ensuring that system behaviors are both correct and secure is a practical necessity [1].

Traditionally, motion planning has focused on finding a collision-free path from a start to a goal position [2]. Classic graph-based algorithms such as Dijkstra's algorithm [3] and A\* [4] are widely used due to their completeness and optimality guarantees. Dijkstra's algorithm computes the shortest path in a weighted graph by exploring all reachable nodes, whereas A\* improves computational efficiency using heuristics to prioritize nodes that are estimated to be closer to the goal.

In continuous or high-dimensional spaces, sampling-based methods, such as Rapidly-exploring Random Trees (RRT) [2] and its optimal variant RRT\* [5] have become standard tools. While RRT finds feasible paths efficiently, it lacks optimality guarantees. In contrast, RRT\* incrementally refines the solution and achieves asymptotic optimality.

However, the planning requirements of modern autonomous systems go far beyond simple reachability or minimum path cost. Agents must fulfill high-level control objectives, such as invariance ("always stay inside region  $G$ "), reach-avoid ("reach  $G$  while avoiding unsafe regions  $O$ "), sequencing ("visit a sequence of waypoints in order"), repeated visits ("visit  $G$  infinitely often"), and until conditions ("stay in  $G$  until visiting  $R$ "). To address this, formal methods, particularly temporal logic-based frameworks, have gained attention [6].

Linear Temporal Logic (LTL) [7] provides a rich and expressive formalism for specifying temporal properties over system trajectories. This expressiveness has led to the development of automata-theoretic planning methods that synthesize correct-by-construction strategies from LTL specifications. An important contribution by De Giacomo and Vardi [8] introduced techniques for translating LTL formulas into automata and composing them with transition systems to generate satisfying executions. Smith et al. [9] extended these ideas to robotic systems and proposed a complete algorithm for synthesizing optimal paths on Weighted Transition Systems (WTS) under LTL constraints.

While much of the early work focused on single-agent systems, recent years have seen growing interest in multi-agent systems with temporal logic objectives [10], [11], [12], [13]. Multi-agent systems can handle complex tasks through coordination, information sharing, and parallel operation. This leads to increased efficiency, robustness to individual agent failures, and the ability to operate effectively in dynamic environments. They are widely used in applications such as surveillance, exploration, and cooperative transportation. However, satisfying high-level temporal goals in such systems introduces new algorithmic and computational challenges, especially with respect to decentralized coordination and task decomposition.

An additional layer of complexity arises from the continuous dynamics of physical systems. Many LTL-based planning methods operate over abstractions, which model agents as finite-state transition systems. While this simplifies reasoning, it abstracts away important constraints related to motion dynamics and actuation. For real-world applications, synthesized plans must be executable by systems governed by differential equations. To bridge this gap, finite abstractions of continuous systems have been proposed, typically through state-space partitioning [14], [15], [16]. These methods discretize the continuous state space into regions, often polytopes, and model transitions between them under conservative approximations of system dynamics. This allows the application of discrete formal synthesis tools to continuous systems. However, constructing abstractions that capture the underlying dynamics and are precise enough for complex task planning remains a nontrivial problem. Multiple approaches have been proposed: Karaman et al. converted the problem to a Mixed-Integer Linear Program [17], while Kress-Gazit et al. [18] developed a framework for synthesizing reactive motion controllers from LTL specifications.

The problem is further amplified in multi-agent systems, where agents may have heterogeneous dynamics, limited communication, and interdependent tasks. Numerous methods have been developed to address coordination under temporal logic constraints [19], [20], [21]. The development of scalable and correct synthesis methods for multi-agent systems with continuous dynamics is an active research area.

Beyond task satisfaction, there is a need to address security in planning and control. Many applications of Cyber-Physical Systems (CPS) involve sensitive operations where agents must conceal information from external observers. For instance, robots may need to access confidential locations, perform private tasks, or avoid revealing mission progress to adversarial intruders. Unfortunately, security concerns are often addressed after system

design, if at all. This introduces the need for integrating security properties directly into the control synthesis process [1].

Opacity has emerged as a key concept for formalizing security in dynamical systems. It ensures that, based on observable outputs, an intruder cannot determine whether a certain secret behavior has occurred. In language-based opacity [22], secret and non-secret paths represent strings in two languages, and a system is considered opaque if all secret paths are confused with non-secret paths by an intruder observing the system [23]. State-based opacity [24], [25] considers a set of secret states and, depending on the type of information that must be concealed, different opacity notions are defined. These two concepts have been widely reviewed and compared [26], [27].

To address more realistic settings, recent work has introduced approximate opacity [28], which accounts for noise and measurement uncertainty in observations. Additional advances have been made in integrating opacity into control frameworks. In [29], opacity requirements are incorporated directly into controller synthesis. A compositional verification method that uses barrier certificates is proposed in [30], enabling scalable opacity checking for interconnected subsystems. For networks of discrete-time switched systems, [31] develops a compositional framework for verifying approximate opacity.

Despite this progress, most existing works on opacity are confined to verification in Discrete Event Systems (DES) [32], [33], [34], with little attention paid to synthesis or integration with continuous dynamics. Some recent studies have begun exploring planning under opacity constraints in DES [35], [36], as well as in multi-agent discrete systems [37]. Others combine LTL control with learning-based methods that incorporate security guarantees [38]. However, there remains a significant gap in designing controllers that satisfy both high-level temporal logic tasks and security specifications in continuous dynamical systems.

## 1.2 Problem Statement and Objectives

The thesis addresses the challenge of synthesizing correct-by-construction and secure-by-construction control strategies for both single-agent and multi-agent systems with affine continuous dynamics. The aim is to develop a planning and control framework that satisfies individual and cooperative LTL task specifications while preserving state-based opacity with respect to external intruders.

Formally, given:

- a multi-agent system modeled either as a Weighted Transition System (WTS) or via continuous-time affine dynamics,
- an LTL formula encoding the system's tasks, and
- a set of secret states or regions,

the goal is to synthesize a control strategy such that:

- The generated system trajectories satisfy the LTL specification.
- The system is secure under various state-based opacity definitions.
- The solution is computationally efficient and scalable to real-world multi-agent settings.

In contrast to approaches that verify opacity after system design, the goal of this work is to integrate security constraints into the planning process, leading to controllers that are secure-by-construction rather than secure-by-verification.

## 1.3 Thesis Contributions

This thesis proposes a unified framework for secure-by-construction planning and control in both discrete and continuous single-agent and multi-agent systems under Linear Temporal Logic (LTL) specifications and state-based opacity constraints.

Three distinct security notions for multi-agent systems are introduced to capture varying intruder capabilities and opacity goals. In Type-A security, the security objective is to maintain path indistinguishability for a subset of agents, named critical agents, from the intruder's perspective. Type-B security introduces a decoy-based strategy, where agents actively cooperate to hide the identity of the ones performing a secret task. Type-C security accounts for bounded delays in the intruder's observations and enables the system to conceal the identity of agents performing secret tasks within a specified time window. These formulations support a variety of security assumptions and make the framework applicable to a wide range of multi-agent settings.

For each security type, a corresponding secure Weighted Transition System (WTS) is constructed. These abstractions come with formal guarantees of correctness - all synthesized paths satisfy both the LTL specification and the defined security constraints - and completeness, that is, all feasible secure paths in the original system are preserved. Moreover, the proposed method achieves lower computational complexity than existing approaches with comparable security guarantees.

The primary contribution of this thesis is the extension of secure planning to continuous single-agent and multi-agent systems with affine dynamics. It constructs secure abstractions of continuous systems and synthesizes continuous trajectories that respect dynamics, LTL specifications and security constraints.

Finally, the proposed framework is validated through many simulation scenarios involving both single-agent and multi-agent systems in various environments. These cases demonstrate the applicability of the approach and show that its theoretical guarantees translate to real-world settings.

## 1.4 Outline of the Thesis

The thesis is structured as follows:

- **Chapter 2** presents the necessary theoretical foundations needed for **Chapter 3** and **Chapter 4**, including LTL syntax and semantics, Weighted Transition Systems (WTS), notions of opacity, continuous-time affine systems, and polytope theory.
- **Chapter 3** introduces the notion of security for single-agent systems, along with three distinct types of security for multi-agent systems: Type-A, Type-B, and Type-C. For each case, it defines an appropriate WTS that includes only secure paths. The Chapter then presents a secure path planning algorithm that ensures satisfaction of LTL specifications for all security types. Finally, it demonstrates the approach in two motivating examples: a single robot navigating within a factory and one of a team of robots coordinating in the same environment.
- **Chapter 4** extends these concepts to continuous systems with affine dynamics. Continuous workspaces are discretized into polytopic regions and serve as discrete abstractions. This allows the application of the secure planning tools developed in **Chapter 3**. An additional assumption is introduced: agents are restricted to move only between the centroids of these polytopes. The resulting continuous trajectories are computed by solving time-optimal control problems.
- **Chapter 5** summarizes the key results and presents directions for future research, including the introduction of decentralized planning, extension to nonlinear and stochastic dynamics, and security against active intruders.

## Chapter 2

# Preliminaries

This Chapter overviews the theoretical foundations necessary for following the methods and results developed later in the thesis. It covers key concepts from Weighted Transition Systems (WTS), Linear Temporal Logic (LTL), security, linear systems and polytopes.

## 2.1 Weighted Transition Systems

A Weighted Transition System (WTS) is a discrete mathematical structure used to model the behavior of systems. It offers a simple yet sufficiently general framework for representing systems operating over discrete or continuous spaces [6].

**Definition 1. Weighted Transition System (WTS)**

A WTS is defined as:

$$\mathcal{T} = (Q, Q_0, \rightarrow, w, \mathcal{AP}, L),$$

where:

- $Q$  is the set of states
- $Q_0 \subseteq Q$  is the set of initial states
- $\rightarrow \subseteq Q \times Q$  is the transition relation defined as: for any  $q, q' \in Q$ ,  $(q, q') \in \rightarrow$  if the transition from  $q$  to  $q'$  is feasible
- $w : Q \times Q \rightarrow \mathbb{R}_+$  is the cost function that assigns to each transition  $(q, q') \in \rightarrow$  a positive weight  $w(q, q')$
- $\mathcal{AP}$  is the set of atomic propositions
- $L : Q \rightarrow 2^{\mathcal{AP}}$  is the labeling function that assigns to each state a set of atomic propositions

In this thesis, WTSs are used as discrete abstractions of the continuous workspaces in which the robots operate. The workspace is partitioned into a finite set of distinct, non-overlapping regions, each corresponding to a state in the WTS. Let  $Q = \{q_1, q_2, \dots, q_p\}$  denote the set of all such states, where each  $q_i \subseteq P$ , and  $P$  is the entire workspace.

The union of all these regions fully cover the workspace, i.e.

$$P = \bigcup_{i=1}^p q_i.$$

Furthermore, the regions are pairwise disjoint:

$$q_i \cap q_j = \emptyset, \forall q_i, q_j \in Q \text{ such that } i \neq j.$$

Paths in a WTS  $\mathcal{T}$  are defined as sequences of states. On a finite-state WTS, paths can be either finite, written as  $\tau = \tau(1)\tau(2) \dots \tau(n)$ , or infinite in a prefix-suffix (lasso) form. The prefix-suffix form is defined as follows:

**Definition 2. Prefix-Suffix Paths**

A path  $\tau$  is in prefix-suffix form if:

$$\tau = \tau(1)\tau(2) \dots \tau(k)[\tau(k+1) \dots \tau(n)]^\omega$$

where:

- The prefix  $\tau(1)\tau(2) \dots \tau(k)$  is a finite sequence, executed only once.
- The suffix  $\tau(k+1) \dots \tau(n)$  is a cycle that repeats indefinitely, as indicated by the superscript  $\omega$ .

The set of all finite paths on  $\mathcal{T}$  is denoted by  $Path(\mathcal{T})$ , and the set of all infinite paths on  $\mathcal{T}$  by  $Path^\omega(\mathcal{T})$ . For a finite path  $\tau$  to belong to  $Path(\mathcal{T})$ , it must satisfy:

- $\tau(1) \in Q_0$ , and
- $(\tau(i), \tau(i+1)) \in \rightarrow, \forall i = 1, 2, \dots, n-1$ .

Similarly, for an infinite path  $\tau$  to be in  $Path^\omega(\mathcal{T})$ , it must hold that:

- $\tau(1) \in Q_0$ ,
- $(\tau(i), \tau(i+1)) \in \rightarrow, \forall i = 1, 2, \dots, n-1$ , and
- $(\tau(n), \tau(k+1)) \in \rightarrow$ , to complete the cycle.

The set of all paths on  $\mathcal{T}$ , including both finite and infinite ones, is denoted by  $Paths(\mathcal{T})$ , where

$$Paths(\mathcal{T}) = Path(\mathcal{T}) \cup Path^\omega(\mathcal{T}).$$

The labeling function  $L$  (see **Definition 1**) can be extended to take paths as input, not just individual states. When applied to a path, this labeling is referred to as a trace. For a finite path  $\tau = \tau(1), \tau(2), \dots, \tau(n)$ , its trace is defined as

$$L(\tau) = L(\tau(1))L(\tau(2)) \dots L(\tau(n))$$

and similarly for infinite paths. The set of all finite traces is denoted by  $Trace(\mathcal{T})$  and the set of all infinite traces by  $Trace^\omega(\mathcal{T})$ .

## 2.2 Linear Temporal Logic

Linear Temporal Logic (LTL) is a formal language used to express temporal properties of systems.

LTL formulas are built over a set of atomic propositions  $\mathcal{AP}$ , using both Boolean and temporal operators. The syntax of LTL can be defined recursively as follows [39]:

$$\phi ::= p \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \mathcal{X}\phi \mid \phi_1 \mathcal{U} \phi_2,$$

where  $p \in \mathcal{AP}$ , and  $\phi, \phi_1, \phi_2$  are LTL formulas. The operators are interpreted as follows:

- $\neg$  (negation): the formula  $\neg\phi$  holds if  $\phi$  does not hold.
- $\wedge$  (conjunction): the formula  $\phi_1 \wedge \phi_2$  holds if both  $\phi_1$  and  $\phi_2$  hold.
- $\mathcal{X}$  (next): the formula  $\mathcal{X}\phi$  holds if  $\phi$  holds at the next discrete time step.
- $\mathcal{U}$  (until): the formula  $\phi_1 \mathcal{U} \phi_2$  holds if  $\phi_1$  holds at every step before  $\phi_2$  holds.

Additional Boolean operators can be derived from the previous:

$$\begin{aligned} \phi_1 \vee \phi_2 &\stackrel{\text{def}}{=} \neg(\neg\phi_1 \wedge \neg\phi_2) \\ \text{true} &\stackrel{\text{def}}{=} p \vee \neg p, \quad \text{false} \stackrel{\text{def}}{=} \neg \text{true} \\ \phi_1 \Rightarrow \phi_2 &\stackrel{\text{def}}{=} \neg\phi_1 \vee \phi_2 \\ \phi_1 \Leftrightarrow \phi_2 &\stackrel{\text{def}}{=} (\phi_1 \Rightarrow \phi_2) \wedge (\phi_2 \Rightarrow \phi_1). \end{aligned}$$

These are interpreted as follows:

- $\vee$  (disjunction): the formula  $\phi_1 \vee \phi_2$  holds if at least one of  $\phi_1$  or  $\phi_2$  holds.
- *true* (true constant): the formula *true* always holds.
- *false* (false constant): the formula *false* never holds.
- $\Rightarrow$  (implication): the formula  $\phi_1 \Rightarrow \phi_2$  holds if either  $\phi_1$  does not hold or  $\phi_2$  holds (or both).
- $\Leftrightarrow$  (equivalence): the formula  $\phi_1 \Leftrightarrow \phi_2$  holds if both formulas hold or both formulas do not hold.

Two additional temporal operators,  $\diamond$  (eventually) and  $\square$  (always), are derived using  $\mathcal{U}$  (until):

$$\begin{aligned}\diamond\phi &\stackrel{\text{def}}{=} \text{true } \mathcal{U} \phi \\ \square\phi &\stackrel{\text{def}}{=} \neg\diamond\neg\phi.\end{aligned}$$

$\diamond$  (eventually) means that  $\phi$  will hold at some point in the future, while  $\square$  (always) means that  $\phi$  holds at every point from now on and forever in the future.

LTL is interpreted over infinite traces. A trace  $r \in \text{Trace}^\omega(\mathcal{T})$  satisfies a formula  $\phi$ , denoted as  $r \models \phi$ , if  $\phi$  holds over  $r$  according to the semantics of LTL.

It has been shown in [40] that for any LTL formula  $\phi$  over a set of atomic propositions  $\mathcal{AP}$ , a Nondeterministic Büchi Automaton (NBA)  $\mathcal{B}_\phi$  can be constructed with input alphabet  $\Sigma = 2^{\mathcal{AP}}$ , such that  $\mathcal{B}_\phi$  accepts exactly the infinite traces that satisfy  $\phi$ . An NBA is defined as follows:

**Definition 3. Nondeterministic Büchi Automaton (NBA) [41]**

An NBA is defined as:

$$\mathcal{B}_\phi = (S, S_0, F, \Sigma, \delta)$$

where:

- $S$  is the finite set of states
- $S_0 \subseteq S$  is the set of initial states
- $F \subseteq S$  is the set of accepting states
- $\Sigma = 2^{\mathcal{AP}}$  is the input alphabet
- $\delta : S \times \Sigma \rightarrow 2^S$  is the transition relation

In this thesis, LTL specifications are used to define tasks for multi-agent systems. To account for task satisfaction, the WTS  $\mathcal{T}$ , which captures the agents' motion, is combined with the NBA  $\mathcal{B}_\phi$ , which captures the satisfaction of the LTL formula. This combination forms a Product Büchi Automaton (PBA), defined as follows:

**Definition 4. Product Büchi Automaton (PBA)**

A PBA is defined as:

$$\mathcal{A}_P = \mathcal{B}_\phi \otimes \mathcal{T} = (S_P, S_{P,0}, F_P, \delta_P, w_P)$$

where:

- $S_P = Q \times S$  is the set of product states
- $S_{P,0} = Q_0 \times S_0$  is the set of initial product states
- $F_P = Q \times F$  is the set of accepting product states
- $\delta_P \subseteq S_P \times S_P$  is the transition relation defined as: for any  $s_p = (q, s) \in S_P$ ,  $s'_p = (q', s') \in S_P$ ,  $(s_p, s'_p) \in \delta_P$  if and only if  $(q, q') \in \rightarrow$  and  $\exists \sigma \in L(q')$  such that  $s' \in \delta(s, \sigma)$
- $w_P : \delta_P \rightarrow \mathbb{R}_+$  is the cost function defined as: for any  $s_p = (q, s) \in S_P$ ,  $s'_p = (q', s') \in S_P$  such that  $(s_p, s'_p) \in \delta_P$ ,  $w_P(s_p, s'_p) = w(q, q')$

There exists a run of the WTS satisfying the formula  $\phi$  if and only if there is an accepting run in the PBA, i.e., one that visits accepting states in  $F_P$  infinitely often [8]. Moreover, if the PBA has at least one accepting run from an initial state, it has at least one accepting run in prefix-suffix form [42].

## 2.3 Security

In this thesis, security refers to the well-established notion of opacity, which characterizes a system's ability to conceal certain secret information about the trajectories of one or more agents. Opacity ensure that an intruder, observing the system indirectly (i.e., through an observation function) cannot infer the system's secret information.

### 2.3.1 Weighted Transition System with Security Features

To formalize opacity for discrete systems, the standard Weighted Transition System (WTS), presented in [Definition 1](#), is extended by incorporating security-related features. [Definition 5](#) augments [Definition 1](#) with:

- an observation function  $H$ , which models what an intruder perceives about the agent's state,
- a set of observations  $Y$ , which is the output space of  $H$ , and
- a set of secret states  $Q_S \subset Q$ , representing the sensitive states that should remain indistinguishable to the intruder.

For opacity to be meaningful, there must exist at least one non-secret state, i.e.  $Q_S \subset Q$ . Otherwise, the system would trivially violate opacity. For the same reason, the initial states must not all be secret, i.e.,  $Q_0 \not\subseteq Q_S$ .

#### **Definition 5. Weighted Transition System (WTS) with Security Features**

*The WTS is defined as:*

$$\mathcal{T} = (Q, Q_0, \rightarrow, w, \mathcal{AP}, L, Y, H, Q_S),$$

where:

- $Q, Q_0, \rightarrow, w, \mathcal{AP}$  and  $L$  are defined as in [Definition 1](#)
- $Y$  is the set of observations
- $H : Q \rightarrow 2^Y$  is the observation function that assigns to each state a set of observations perceived by the intruder
- $Q_S \subset Q$  is the set of secret states

To reason about the intruder's knowledge, it is possible to distinguish between secret and non-secret paths and define how system executions are observed externally.

**Definition 6. Secret Path**

A (finite or infinite) path  $\tau$  on  $\mathcal{T}$  is a secret path if it contains at least one secret state, i.e., if there exists  $i$  such that  $\tau(i) \in Q_S$ .

**Definition 7. Non-Secret Path**

A (finite or infinite) path  $\tau$  on  $\mathcal{T}$  is a non-secret path if it contains only non-secret states, i.e.,  $\tau(i) \in Q \setminus Q_S$  for all  $i$ .

Just as the labeling function  $L$  can be extended to paths, so can the observation function  $H$ . When  $H$  is applied to a path, it returns the external path, that is the sequence of observations corresponding to the path.

**Definition 8. External Path**

Given a (finite or infinite) path  $\tau = \tau(1)\tau(2)\dots$  on  $\mathcal{T}$ , the external path is defined as the sequence of observations  $H(\tau) = H(\tau(1))H(\tau(2))\dots$ . An external path represents what an intruder perceives while the agent follows the path  $\tau$ .

### 2.3.2 Language-Based Opacity

Opacity can be categorized into language-based and state-based forms, as discussed in [1]. Language-based opacity refers to the concept of strong opacity presented in [23].

**Definition 9. Language-Based Opacity [23]**

Let  $\mathcal{T} = (Q, Q_0, \rightarrow, w, \mathcal{AP}, L, Y, H, Q_S)$ . Let  $Path_s \subseteq Paths(\mathcal{T})$  denote the set of all secret paths on  $\mathcal{T}$  and  $Path_{ns} \subseteq Paths(\mathcal{T})$  the set of all non-secret paths on  $\mathcal{T}$ . Then,  $\mathcal{T}$  is said to be language-based opaque with respect to  $Path_s$ ,  $Path_{ns}$  and  $H$  if, for every secret path  $\tau \in Path_s$ , there exist a non-secret path  $\tau' \in Path_{ns}$  such that  $H(\tau) = H(\tau')$ .

In other words, an intruder observing the system cannot determine whether a secret path has been followed, as there always exists an indistinguishable non-secret path.

Language-based opacity is a very strong form of opacity and its satisfaction implies the satisfaction of all the variants of state-based opacity discussed in the following section.

### 2.3.3 State-Based Opacity

Let  $\mathcal{T} = (Q, Q_0, \rightarrow, w, \mathcal{AP}, L, Y, H, Q_S)$  be a WTS. The following state-based notions of opacity are considered:

**Definition 10. Initial-State Opacity [43]**

*System  $\mathcal{T}$  is initial-state opaque if, for every path  $\tau = \tau(1)\tau(2)\dots \in \text{Paths}(\mathcal{T})$  such that the initial state  $\tau(1) \in Q_S$ , there exists another path  $\tau' = \tau'(1)\tau'(2)\dots\tau'(n) \in \text{Path}(\mathcal{T})$  with  $\tau'(1) \notin Q_S$  such that  $H(\tau) = H(\tau')$ .*

**Definition 11. Current-State Opacity [24]**

*System  $\mathcal{T}$  is current-state opaque if, for every path  $\tau = \tau(1)\tau(2)\dots\tau(n) \in \text{Path}(\mathcal{T})$  such that  $\tau(n) \in Q_S$ , there exists another path  $\tau' = \tau'(1)\tau'(2)\dots\tau'(n) \in \text{Path}(\mathcal{T})$  with  $\tau'(n) \notin Q_S$  such that  $H(\tau) = H(\tau')$ .*

**Definition 12. Infinite-Step Opacity [33]**

*System  $\mathcal{T}$  is infinite-step opaque if, for every path*

$$\tau = \tau(1)\tau(2)\dots\tau(n)\dots\tau(n+k) \in \text{Path}(\mathcal{T})$$

*such that  $\tau(n) \in Q_S$  for some  $k \in \mathbb{N}$ , there exists another path  $\tau' = \tau'(1)\tau'(2)\dots\tau'(n)\dots\tau'(n+k) \in \text{Path}(\mathcal{T})$  with  $\tau'(n) \notin Q_S$  such that  $H(\tau) = H(\tau')$ .*

**Definition 13. K-Step Opacity [32]**

*System  $\mathcal{T}$  is K-step opaque if, for every path  $\tau = \tau(1)\tau(2)\dots\tau(n)\dots\tau(n+k) \in \text{Path}(\mathcal{T})$  such that  $\tau(n) \in Q_S$  with  $k \leq K$ , there exists another path  $\tau' = \tau'(1)\tau'(2)\dots\tau'(n)\dots\tau'(n+k) \in \text{Path}(\mathcal{T})$  with  $\tau'(n) \notin Q_S$  such that  $H(\tau) = H(\tau')$ .*

In this thesis, an alternative security property is formalized in **Definition 14**. This definition ensures that the intruder, observing only external paths through an observation function  $H$ , remains uncertain about whether a secret

state has been visited. Specifically, for a path to be secure, there must exist an alternative path producing the same sequence of observations while avoiding visiting secret states at the same time as the real path. Formally:

**Definition 14. Security**

System  $\mathcal{T}$  is secure if, for every path  $\tau \in Paths(\mathcal{T})$ , there exists a path  $\tau' \in Paths(\mathcal{T})$  such that:

- The two paths share the same external path, i.e.,  $H(\tau) = H(\tau')$ , and
- $\forall i = 1, \dots, n$ : if  $\tau(i) \in Q_S$ , then  $\tau'(i) \notin Q_S$ .

Specifically:

- If  $\tau$  is a finite path, i.e.,  $\tau \in Path(\mathcal{T})$ , of the form  $\tau = \tau(1)\tau(2) \dots \tau(n)$ , then  $\tau' = \tau'(1)\tau'(2) \dots \tau'(n)$ .
- If  $\tau$  is an infinite prefix-suffix path, i.e.  $\tau \in Path^\omega(\mathcal{T})$ , of the form

$$\tau = \tau(1)\tau(2) \dots \tau(k)[\tau(k+1) \dots \tau(n)]^\omega,$$

then

$$\tau' = \tau'(1)\tau'(2) \dots \tau'(k)[\tau'(k+1) \dots \tau'(n)]^\omega.$$

This notion of security is stronger than all previously defined types of state-based opacity and is applicable to both finite and infinite prefix-suffix paths. However, it is weaker than language-based opacity. Language-based opacity requires that, for every secret path, a non-secret path that produces identical observation exists. In contrast, the definition in **Definition 14** only requires the existence of an indistinguishable path  $\tau'$  that avoids visiting secret states at the same time steps as  $\tau$ . This means that  $\tau'$  may still contain secret states, provided they occur at different time steps.

## 2.4 Linear Systems

A linear system is defined by a set of ordinary differential equations of the form

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}, \quad (2.1)$$

where  $\mathbf{x} \in \mathbb{R}^N$  is the state vector,  $\mathbf{u} \in \mathbb{R}^m$  is the input vector, and  $\mathbf{A} \in \mathbb{R}^{N \times N}$  and  $\mathbf{B} \in \mathbb{R}^{N \times m}$  are constant matrices. An extension of this model includes an additional constant vector  $\mathbf{b} \in \mathbb{R}^N$ :

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} + \mathbf{b}.$$

This formulation represents an affine system, rather than a purely linear one.

A special class of linear systems is the single integrator system, which is described by:

$$\dot{\mathbf{x}} = \mathbf{u}, \quad \mathbf{x}, \mathbf{u} \in \mathbb{R}^N. \quad (2.2)$$

This can be expressed in the standard continuous-time state-space form of [Equation 2.1](#) by setting:

$$\mathbf{A} = \mathbf{0}_{N \times N}, \quad \mathbf{B} = \mathbf{I}_{N \times N}.$$

For the one-dimensional case, this simplifies to:

$$A = 0, \quad B = 1.$$

## 2.5 Polytopes

A polytope is a bounded convex set contained in  $\mathbb{R}^N$ . It is said to be full-dimensional if it has a non-empty interior in  $\mathbb{R}^N$ , i.e., it spans in all dimensions of  $\mathbb{R}^N$ . Full-dimensional polytopes in  $\mathbb{R}^N$  can be constructed as the convex hull of at least  $N + 1$  affinely independent points in  $\mathbb{R}^N$ .

Let  $P$  be a full-dimensional polytope in  $\mathbb{R}^N$  with a finite set of vertices  $\{v_1, \dots, v_M\}$ , where  $v_i \in \mathbb{R}^N, i = 1, \dots, M$ . Then  $P$  can be expressed as the convex hull of these vertices:

$$P = \text{conv}(\{v_1, \dots, v_M\}),$$

which is known as the vertex representation (V-representation) of the polytope.

Alternatively,  $P$  can be described as the intersection of finitely many closed half-spaces:

$$P = \{\mathbf{x} \in \mathbb{R}^N \mid \mathbf{H}\mathbf{x} + \mathbf{g} \leq 0\},$$

where  $H \in \mathbb{R}^{K \times N}$  and  $g \in \mathbb{R}^K$ . This formulation is referred to as the half-space representation (H-representation) of the polytope.

Efficient algorithms exist for the translation between the V-representation and H-representation.

## Chapter 3

# Secure-by-Construction Optimal Path Planning for Discrete Systems with LTL Specifications

This Chapter introduces a framework for finding optimal plans under security constraints in both single-agent and multi-agent systems. The framework consists of four main parts:

- **Optimal Path Planning under LTL Specifications:** Develops an algorithm to find infinite paths that satisfy a given Linear Temporal Logic (LTL) formula while ensuring optimality.
- **Security Analysis for Single-Agent Systems:** Investigates security for individual agents navigating a WTS.
- **Security Extension to Multi-Agent Systems:** Expands the security analysis to systems involving multiple agents, defining three distinct security notions.
- **Optimal Path Planning under Security Constraints and LTL Tasks:** Integrates optimality and security into a unified path planning algorithm, ensuring both objectives are satisfied.

### 3.1 Optimal Path Planning under LTL Specifications

**Algorithm 1** presents an optimal path planning algorithm designed to find the shortest path in prefix-suffix form, as defined in **Definition 2**, on a Weighted Transition System (WTS). The algorithm guarantees satisfaction of a given LTL formula, provided such a path exists. The path starts from an initial state  $q_0 \in Q_0$ , where  $Q_0$  denotes the set of initial states of the WTS. The resulting optimal path  $\tau^*$  is:

$$\tau^* = \tau^*(1)\tau^*(2) \dots \tau^*(k)[\tau^*(k+1) \dots \tau^*(n)]^\omega$$

Its total cost  $J$  is given by:

$$J = \alpha \cdot J_{\text{prefix}} + (1 - \alpha) \cdot J_{\text{suffix}}, \quad \alpha \in [0, 1], \quad (3.1)$$

where  $w$  is the weight function as defined in **Definition 1**, and

$$J_{\text{prefix}} = w(\tau^*(1)\tau^*(2) \dots \tau^*(k+1)) = \sum_{i=1}^k w(\tau^*(i), \tau^*(i+1)),$$

and

$$\begin{aligned} J_{\text{suffix}} &= w(\tau^*(k+1) \dots \tau^*(n)\tau^*(k+1)) \\ &= \sum_{i=k+1}^{n-1} w(\tau^*(i), \tau^*(i+1)) + w(\tau^*(n), \tau^*(k+1)). \end{aligned}$$

The algorithm ensures optimality by minimizing this total path cost. The optimal path planning algorithm consists of five steps:

- **Büchi Automaton Construction:** The algorithm begins by constructing the Büchi automaton  $\mathcal{B}_\phi$  from the given LTL formula  $\phi$  (see *buildBüchiAutomaton* in **Algorithm 1**).
- **Product Automaton Construction:** Next, the product Büchi automaton  $\mathcal{A}_P$  is built by combining the Büchi automaton  $\mathcal{B}_\phi$  with the WTS  $\mathcal{T}$ . This step is symbolized by the function *buildProductBüchiAutomaton* in **Algorithm 1**.
- **Path and Cost Initialization:** The optimal path  $\tau^*$  and its corresponding minimum cost  $J_{\min}$  are initialized.

- **Prefix-Suffix Path Computation:** The algorithm iterates over all accepting states  $q_{acc} \in Q_{acc}$  in the product Büchi automaton. The shortest prefix path from any initial state  $q_0 \in Q_0$  to the current accepting state is first computed using Dijkstra's algorithm [3]. Then, another application of Dijkstra's algorithm finds the shortest suffix (cycle) path, using  $q_{acc}$  as both the starting state and the goal state.
- **Path Selection:** The shortest prefix-suffix path passing through each  $q_{acc} \in Q_{acc}$  is formed by concatenating the computed prefix and suffix paths. The total cost is evaluated using Equation 3.1. If the newly found path is shorter than the best one found so far, the minimum cost and the optimal path are updated accordingly. Once all accepting states in  $Q_{acc}$  have been considered, the algorithm returns the optimal path and its corresponding minimum cost.

---

**Algorithm 1** Optimal Path Planning

---

**Input:** LTL formula  $\phi$ , WTS  $\mathcal{T}$ , weight factor  $\alpha$

**Output:** optimal path  $\tau^*$ , minimum path cost  $J_{\min}$

- 1:  $\mathcal{B}_\phi \leftarrow \text{buildBüchiAutomaton}(\phi)$
  - 2:  $\mathcal{A}_P \leftarrow \text{buildProductBüchiAutomaton}(\mathcal{T}, \mathcal{B}_\phi)$
  - 3:  $\tau^* \leftarrow \emptyset$
  - 4:  $J_{\min} \leftarrow \infty$
  - 5: **for** each accepting state  $q_{acc} \in Q_{acc}$  **do**
  - 6:     Compute shortest prefix path  $\pi_{\text{prefix}}$  from any initial state in  $Q_0$  to  $q_{acc}$
  - 7:     Compute shortest cycle  $\pi_{\text{cycle}}$  starting from  $q_{acc}$  and returning to  $q_{acc}$
  - 8:     Compute total cost  $J_{\text{total}} = \alpha \cdot J_{\text{prefix}} + (1 - \alpha) \cdot J_{\text{suffix}}$
  - 9:     **if**  $J_{\text{total}} < J_{\min}$  **then**
  - 10:          $J_{\min} \leftarrow J_{\text{total}}$
  - 11:          $\tau^* \leftarrow \pi_{\text{prefix}} \oplus \pi_{\text{cycle}}$
  - 12: **return**  $\tau^*, J_{\min}$
-

## 3.2 Security in Single-Agent Systems

In addition to the optimal path planning problem under Linear Temporal Logic (LTL) specifications, security constraints are introduced. The concept of security is first explored in the context of single-agent systems. The underlying intuition is outlined below:

### *Intruder & Security*

A malicious intruder aims to determine whether the agent has visited a secret state at any point along its trajectory. The intruder does not have direct access to the agent's exact location; instead, they can only get observations associated with that location through the observation function defined in [Definition 5](#). As a result, the intruder is not able to distinguish between states that share the same observation. The intruder is assumed to have knowledge of the system's secret and initial locations, as well as the system's transition relation.

### 3.2.1 Definitions

In this Subsection, all necessary definitions related to security in single-agent systems are presented. The discussion begins with the abstraction of the agent's motion, namely the Weighted Transition System (WTS). Following this, the desired security property is formally defined for discrete systems.

To enforce this property, two additional WTS constructions are introduced. First, the Twin-Weighted Transition System (Twin-WTS), which was introduced in [\[36\]](#) and captures all pairs of states that are indistinguishable from the intruder's perspective, i.e., those that yield identical observations. Second, the Secure Twin-Weighted Transition System (Twin-sWTS), which is derived from the Twin-WTS by retaining only the state pairs that satisfy the security condition formalized in [Definition 14](#).

Paths in the Twin-sWTS can, by construction, maintain plausible deniability about visiting secret states. For every such path, there exists a corresponding copy path that produces the same sequence of observations.

#### 3.2.1.1 System Model

The agent's motion is modeled by a Weighted Transition System (WTS) with security-related attributes, as formalized previously in [Definition 5](#).

### 3.2.1.2 Security Property

As discussed in [Chapter 2](#), the intruder does not know the agent's exact path but instead perceives a sequence of observations via the observation function  $H$ , forming what was defined as the external path in [Definition 8](#).

In this setting, the security property (introduced in [Definition 14](#)) requires that for any given path, there exists an alternative path producing the same sequence of observations, while avoiding secret states at the same time steps. This ensures that an intruder cannot infer, at any time step, whether a secret state has been visited.

### 3.2.1.3 Construction of Secure WTS

The Twin-Weighted Transition System (Twin-WTS) was proposed in [\[36\]](#) in order to keep track of all the pairs of states of the original WTS that produce the same external path for the intruder. It is defined as follows:

**Definition 15. Twin-Weighted Transition System (Twin-WTS) [\[36\]](#)**

Given a WTS

$$\mathcal{T} = (Q, Q_0, \rightarrow, w, \mathcal{AP}, L, Y, H, Q_S),$$

as defined in [Definition 5](#), its Twin-WTS is a new WTS

$$\mathcal{V} = (X, X_0, \rightarrow_V, w_V, \mathcal{AP}, L_V, Y, H_V, Q_S),$$

where:

- $X = \{(q_1, q_2) \in Q \times Q \mid H(q_1) = H(q_2)\}$  is the set of states
- $X_0 = \{(q_1, q_2) \in Q_0 \times Q_0 \mid H(q_1) = H(q_2)\}$  is the set of initial states
- $\rightarrow_V \subseteq X \times X$  is the transition relation defined as: for any  $x = (q_1, q_2) \in X$  and  $x' = (q'_1, q'_2) \in X$ ,  $(x, x') \in \rightarrow_V$  if the following hold:
  - $(q_1, q'_1) \in \rightarrow$
  - $(q_2, q'_2) \in \rightarrow$
  - $H(q'_1) = H(q'_2)$
- $w_V : X \times X \rightarrow \mathbb{R}_+$  is the cost function defined as: for any  $x = (q_1, q_2) \in X$  and  $x' = (q'_1, q'_2) \in X$  such that  $(x, x') \in \rightarrow_V$ ,  $w_V(x, x') = w(q_1, q'_1)$
- $L_V : X \rightarrow 2^{\mathcal{AP}}$  is the labeling function defined as: for any  $x = (q_1, q_2) \in X$ ,  $L_V(x) = L(q_1)$

- $H_V : X \rightarrow Y$  is the observation function defined as: for any  $x = (q_1, q_2) \in X$ ,  $H_V(x) = H(q_1)$
- $\mathcal{AP}$ ,  $Y$  and  $Q_S$  are directly inherited from  $\mathcal{T}$

States of the Twin-WTS are referred to as twin states, where each consists of two components: the real state and its corresponding copy state. For instance, in the twin state  $(q_1, q_2)$ ,  $q_1$  represents the real state, while  $q_2$  denotes the copy state.

A Secure Twin-WTS (Twin-sWTS) is obtained from a Twin-WTS by removing all states  $(q_1, q_2)$  for which both  $q_1$  and  $q_2$  belong to  $Q_S$ . Additionally, all transitions involving these removed states are eliminated. Formally, it is defined as follows:

**Definition 16. Secure Twin-Weighted Transition System (Twin-sWTS)**

Given a Twin-WTS

$$\mathcal{V} = (X, X_0, \rightarrow_V, w_V, \mathcal{AP}, L_V, Y, H_V, Q_S),$$

its Twin-sWTS is a new WTS

$$\mathcal{V}_s = (X_s, X_{0,s}, \rightarrow_s, w_s, \mathcal{AP}, L_s, Y, H_s, Q_S),$$

where:

- $X_s = X \setminus \{(q_1, q_2) \mid q_1 \in Q_S \text{ and } q_2 \in Q_S\}$  is the set of states
- $X_{0,s} = X_s \cap X_0$  is the set of initial states
- $\rightarrow_s \subseteq X_s \times X_s$  is the transition relation defined as: for any  $x_s, x'_s \in X_s$ ,  $(x_s, x'_s) \in \rightarrow_s$  if and only if  $(x_s, x'_s) \in \rightarrow_V$
- $w_s : X_s \times X_s \rightarrow \mathbb{R}_+$  is the cost function inherited from  $\mathcal{V}$  and defined as: for any  $x_s, x'_s \in X_s$  such that  $(x_s, x'_s) \in \rightarrow_s$ ,  $w_s(x_s, x'_s) = w_V(x_s, x'_s)$
- $L_s : X_s \rightarrow 2^{\mathcal{AP}}$  is the labeling function inherited from  $\mathcal{V}$  and defined as: for any  $x_s \in X_s$ ,  $L_s(x_s) = L_V(x_s)$
- $H_s : X_s \rightarrow Y$  is the observation function inherited from  $\mathcal{V}$  and defined as: for any  $x_s \in X_s$ ,  $H_s(x_s) = H_V(x_s)$
- $\mathcal{AP}$ ,  $Y$  and  $Q_S$  are directly inherited from  $\mathcal{V}$

For **Definition 17**, **Theorem 1**, and **Theorem 2**, consider a WTS

$$\mathcal{T} = (Q, Q_0, \rightarrow, w, \mathcal{AP}, L, Y, H, Q_S).$$

Let

$$\mathcal{V} = (X, X_0, \rightarrow_V, w_V, \mathcal{AP}, L_V, Y, H_V, Q_S),$$

be its corresponding Twin-WTS, and let

$$\mathcal{V}_s = (X_s, X_{0,s}, \rightarrow_s, w_s, \mathcal{AP}, L_s, Y, H_s, Q_S)$$

be its corresponding Twin-sWTS.

A function is needed to project the twin states of the Twin-sWTS back onto their real states in the WTS.

**Definition 17. Path projection from Twin-sWTS onto WTS**

Two projection functions  $\Pi, \Pi' : X_s \rightarrow Q$  are defined as follows: for any  $x_s = (q_1, q_2) \in X_s$ ,  $\Pi(x_s) = q_1$  and  $\Pi'(x_s) = q_2$ .

These functions extend naturally to paths. For a path  $x_s \dots x'_s = (q_1, q_2) \dots (q'_1, q'_2)$  in the Twin-sWTS, the projected paths in the WTS are

$$\Pi(x_s \dots x'_s) = q_1 \dots q'_1 \text{ and } \Pi'(x_s \dots x'_s) = q_2 \dots q'_2.$$

### 3.2.2 Security Guarantees

**Theorem 1** is used to prove that projecting any path from the Twin-sWTS onto the WTS will never result in a non-secure path. Formally:

**Theorem 1.** *The projection of any path in the finite-state Twin-sWTS  $\mathcal{V}_s$  onto the WTS  $\mathcal{T}$  is secure.*

*Proof.* All feasible paths in a finite-state WTS are either finite or in prefix-suffix form. Therefore, **Theorem 1** will be proven separately for finite paths and for infinite prefix-suffix paths.

Consider first a finite path  $\sigma$  in  $\mathcal{V}_s$ , given by

$$\sigma = \sigma(1)\sigma(2) \dots \sigma(n) = \tau\tau' = (\tau(1), \tau'(1))(\tau(2), \tau'(2)) \dots (\tau(n), \tau'(n)).$$

By applying the projection function, a corresponding finite path in  $\mathcal{T}$  is obtained:

$$\tau = \Pi(\sigma) = \tau(1)\tau(2) \dots \tau(n).$$

By the definition of  $V_s$ , the copy path

$$\tau' = \tau'(1)\tau'(2) \dots \tau'(n)$$

produces the same sequence of observations as  $\tau$ , i.e.,  $H(\tau) = H(\tau')$ .

Moreover, for all indices  $i \in \{1, \dots, n\}$  such that  $\tau(i) \in Q_S$ , it must hold that  $\tau'(i) \notin Q_S$ . Otherwise  $\mathcal{V}_s$  would contain the state  $\sigma(i) = (\tau(i), \tau'(i))$  with both  $\tau(i)$  and  $\tau'(i)$  in  $Q_S$ . This contradicts the construction of  $\mathcal{V}_s$ , as such states were removed from the Twin-WTS  $\mathcal{V}$  to form  $\mathcal{V}_s$ . Hence, the path  $\tau$  satisfies the security condition specified in **Definition 14**.

Extending the argument to infinite prefix-suffix paths, the projection function  $\Pi$  is applied to obtain a corresponding infinite prefix-suffix path in  $\mathcal{T}$ . The proof follows the same reasoning as for finite paths.  $\square$

**Theorem 2** ensures that no secure path is excluded from the planning algorithm's search. This is particularly important when seeking an optimal plan, as it guarantees that the optimal solution is preserved during the construction of  $\mathcal{V}_s$ .

**Theorem 2.** *Let  $\tau = \tau(1)\tau(2) \dots$  be a secure path in the WTS  $\mathcal{T}$ . Then, there exists at least one path  $\sigma = \sigma(1)\sigma(2) \dots$  in the Twin-sWTS  $\mathcal{V}_s$  such that  $\Pi(\sigma) = \tau$ .*

*Proof.* Since  $\tau$  is secure, by definition there exists at least one copy path  $\tau' = \tau'(1)\tau'(2) \dots$  such that:

1.  $H(\tau) = H(\tau')$ , and
2. for all  $i$ , either  $\tau(i) \notin Q_S$  or  $\tau'(i) \notin Q_S$ .

Define  $\sigma(i) = (\tau(i), \tau'(i))$  for all  $i$ . Each  $\sigma(i) = (\tau(i), \tau'(i))$  belongs to the Twin-WTS  $\mathcal{V}$ . Moreover, it is retained in the Twin-sWTS  $\mathcal{V}_s$  since either  $\tau(i) \notin Q_S$  or  $\tau'(i) \notin Q_S$ .

Furthermore, since  $(\tau(i), \tau(i+1)) \in \rightarrow$  and  $(\tau'(i), \tau'(i+1)) \in \rightarrow$ , it follows that  $(\sigma(i), \sigma(i+1)) \in \rightarrow_s$ , making  $\sigma$  a valid path in  $\mathcal{V}_s$ .

Thus,  $\sigma$  is a path in the Twin-sWTS  $\mathcal{V}_s$  and satisfies  $\Pi(\sigma) = \tau$ .  $\square$

### 3.3 Security in Multi-Agent Systems

In multi-agent systems, the concept of security becomes significantly more complex, allowing for multiple interpretations depending on the system's objectives and the intruder's capabilities. In this thesis, three distinct security notions are introduced. Each of these security definitions assumes the presence of a passive intruder trying to extract critical information about the agents' paths.

#### 3.3.1 Type-A Security

Type-A security extends the single-agent security framework to multi-agent settings:

***Type-A Security***

A subset of agents, called critical agents, is denoted by  $A_c \subseteq A$ . Type-A security focuses on whether any critical agent has visited a secret state. In this setting, the intruder is assumed to observe each agent individually but focuses exclusively on the behavior of the critical agents. Additionally, the intruder is assumed to have knowledge of the system's secret and initial locations, as well as the system's transition relation.

##### 3.3.1.1 System Model

The single-agent transition systems described above can be generalized and extended to handle multi-agent systems. Consider a multi-agent system with  $m$  agents. The set containing all  $m$  agents is denoted as  $A = \{1, \dots, m\}$ . The motion of each agent  $i \in A$  is represented by a Weighted Transition System (WTS)

$$\mathcal{T}^i = (Q^i, Q_0^i, \rightarrow_i, w_i, \mathcal{AP}_i, L_i, Y_i, H_i, Q_S^i),$$

as defined in [Definition 5](#). Note that agents are not required to share the same set of secret states.

By composing the individual WTSs of all  $m$  agents, a Global Weighted Transition System is obtained, defined as follows:

**Definition 18. Global Type-A WTS**

A Global WTS is defined as:

$$\mathcal{T}_g^A = \bigotimes_{i=1}^m \mathcal{T}^i = (P, P_0, \rightarrow_g, w_g, \mathcal{AP}_g, L_g, A, A_c, Y_g, H_g, Q_{S,g})$$

where:

- $P = Q^1 \times Q^2 \times \dots \times Q^m$  is the set of global states
- $P_0 = Q_0^1 \times Q_0^2 \times \dots \times Q_0^m \subseteq P$  is the set of initial global states
- $\rightarrow_g \subseteq P \times P$  is the transition relation defined as: for any  $p = (p^1, p^2, \dots, p^m)$ ,  $p' = (p'^1, p'^2, \dots, p'^m) \in P$ ,  $(p, p') \in \rightarrow_g$  if  $(p^i, p'^i) \in \rightarrow_i, \forall i = 1, \dots, m$
- $w_g : P \times P \rightarrow \mathbb{R}_+$  is the cost function defined as: for any  $p = (p^1, p^2, \dots, p^m)$ ,  $p' = (p'^1, p'^2, \dots, p'^m) \in P$  such that  $(p, p') \in \rightarrow_g$ ,  $w_g(p, p') = \sum_{i=1}^m w_i(p^i, p'^i)$
- $\mathcal{AP}_g = 2^{\mathcal{AP}_1} \times \dots \times 2^{\mathcal{AP}_m}$  is the set of atomic propositions
- $L_g : P \rightarrow \mathcal{AP}_g$  is the labeling function defined as: for any  $p = (p^1, p^2, \dots, p^m) \in P$ ,  $L_g(p) = (L_1(p^1), \dots, L_m(p^m))$
- $A$  is the set of agents
- $A_c \subseteq A$  is the set of critical agents
- $Y_g = Y_1 \times \dots \times Y_m$  is the set of observations
- $H_g : P \rightarrow Y_g$  is the observation function defined as: for any  $p = (p^1, p^2, \dots, p^m) \in P$ ,  $H_g(p) = (H_1(p^1), \dots, H_m(p^m))$
- $Q_{S,g} = Q_S^1 \times \dots \times Q_S^m$  is the set of secret states

The superscript  $A$  in  $\mathcal{T}_g^A$  signifies Type-A security. The states of the Global WTS, denoted as  $p = (p^1, p^2, \dots, p^m) \in P$ , are referred to as global states, while each  $p^i, i = 1, \dots, m$ , is called a local state.

While in the single-agent case each state is directly associated with a set of atomic propositions and a set of observations, the multi-agent setting demands a different approach. The labeling function  $L_g$  captures properties relevant to temporal logic specifications. For each agent  $i \in A$ , the label at a local state

$p^i$  is defined independently and not merged with those of other agents. This separation is essential, as the global LTL formula includes individual (agent-specific) predicates that must be evaluated individually. Similarly, in each global state  $p$ , each agent's observation remains distinct. This is consistent with the definition of the critical agent set  $A_c$ : only the behavior of critical agents is subject to security constraints.

### 3.3.1.2 Security Property

The security property is formally presented in [Definition 19](#).

#### **Definition 19. Global Path Type-A Security**

A global path  $\tau$  in  $\mathcal{T}_g^A$ , where a local path  $\tau^j = (\tau^j(1), \tau^j(2), \dots, \tau^j(n))$  is defined for each agent  $j \in A$ , is secure if there exists a global path  $\tau'$  in  $\mathcal{T}_g^A$  such that:

- $H(\tau^j) = H(\tau'^j), \forall j \in A_c$ , and
- $\forall i = 1, \dots, n$ , and  $\forall j \in A_c$ : if  $\tau^j(i) \in Q_S^j$ , then  $\tau'^j(i) \notin Q_S^j$ .

### 3.3.1.3 Construction of Secure WTS

The Global Twin-WTS, which is the direct multi-agent counterpart of the Twin-WTS, is constructed based on a slightly modified principle. Specifically:

- **For critical agents:** Copy paths must produce the same external path from the intruder's perspective. This ensures plausible deniability about visiting secret states.
- **For non-critical agents:** No such restriction applies, as their locations are assumed to be irrelevant to the intruder. This means that their paths do not need to be indistinguishable from alternative paths, even if they pass through secret states.

Our primary goal is to ensure that whenever a critical agent  $i$  follows a path  $\tau$  passing through a secret state, i.e.,  $\tau(j) = p^i \in Q_S^i$ , there exists a copy path  $\tau'$  that, at the same time, passes through a non-secret state, i.e.,  $\tau'(j) = p^i \in Q^i \setminus Q_S^i$ , such that the two paths remain indistinguishable to the intruder. This guarantees that if a critical agent ever enters a secret state, there will always be an alternative indistinguishable path that, at the same time, passes through a non-secret state, preserving Type-A security.

Applying the same condition to non-critical agents is both unnecessary and computationally inefficient. Since the intruder ignores their observations, introducing copy states for non-critical agents would only enlarge the state space and increase computational complexity without contributing to Type-A security.

Therefore, the Global Twin-WTS consists of all states of the form  $(p_1, p_2) = ((p_1^1, p_1^2, \dots, p_1^m), (p_2^1, p_2^2, \dots, p_2^m))$ , where  $p_1, p_2 \in P$ , such that:

- **For critical agents:**  $H_i(p_1^i) = H_i(p_2^i), \forall i \in A_c$
- **For non-critical agents:**  $p_1^i = p_2^i, \forall i \in A \setminus A_c$

To illustrate, consider a system with three agents, where the set of critical agents  $A_c = \{1, 2\}$  and the set of non-critical agents is  $A \setminus A_c = \{3\}$ . Given a global state  $p = (p^1, p^2, p^3)$ , any valid copy state  $p' = (p'^1, p'^2, p'^3)$  must satisfy the following:

- $H_1(p^1) = H_1(p'^1)$  (indistinguishability for critical agent 1)
- $H_2(p^2) = H_2(p'^2)$  (indistinguishability for critical agent 2)
- $p^3 = p'^3$  (non-critical agent 3 is at the same state in real and copy paths)

The formal definition of the Global Twin-WTS is given in [Definition 20](#).

**Definition 20. Global Twin-WTS**

*Given a Global WTS*

$$\mathcal{T}_g^A = (P, P_0, \rightarrow_g, w_g, \mathcal{AP}_g, L_g, A, A_c, Y_g, H_g, Q_{S,g}),$$

*its Global Twin-WTS is a new WTS*

$$\mathcal{V} = (X, X_0, \rightarrow_V, w_V, \mathcal{AP}_g, L_V, A, A_c, Y_g, H_V, Q_{S,g}),$$

*where:*

- $X \subseteq P \times P$  is the set of states
- $X_0 \subseteq P_0 \times P_0$  is the set of initial states defined as: for any  $p_1 = (p_1^1, p_1^2, \dots, p_1^m) \in P_0$  and  $p_2 = (p_2^1, p_2^2, \dots, p_2^m) \in P_0$ ,  $(p_1, p_2) \in X_0$  if and only if the following hold:
  - $H_i(p_1^i) = H_i(p_2^i), \forall i \in A_c$

- $p_1^i = p_2^i, \forall i \in A \setminus A_c$
- $\rightarrow_V \subseteq X \times X$  is the transition relation defined as: for any  $x = (p_1, p_2) \in X$  and  $x' = (p'_1, p'_2) \in X$ ,  $(x, x') \in \rightarrow_V$  if the following hold:
  - $(p_1, p'_1) \in \rightarrow_g$
  - $(p_2, p'_2) \in \rightarrow_g$
  - $H_i(p_1) = H_i(p'_1), \forall i \in A_c$
  - $p_1^i = p_2^i, \forall i \in A \setminus A_c$
- $w_V : X \times X \rightarrow \mathbb{R}_+$  is the cost function defined as: for any  $x = (p_1, p_2) \in X$  and  $x' = (p'_1, p'_2) \in X$  such that  $(x, x') \in \rightarrow_V$ ,  $w_V(x, x') = w_g(p_1, p'_1)$
- $L_V : X \rightarrow \mathcal{AP}_g$  is the labeling function defined as: for any  $x = (p_1, p_2) \in X$ ,  $L_V(x) = L_g(p_1)$
- $H_V : X \rightarrow Y_g$  is the observation function defined as: for any  $x = (p_1, p_2) \in X$ ,  $H_V(x) = H_g(p_1)$
- $\mathcal{AP}_g, A, A_c, Y_g$  and  $Q_{S,g}$  are directly inherited from  $\mathcal{T}_g^A$

The Global Secure Twin-WTS (Global Twin-sWTS) is constructed similarly to its single-agent counterpart, the Twin-sWTS. Specifically, starting from the Global Twin-WTS, a twin global state  $x = (p_1, p_2)$  is retained if, for each critical agent  $i \in A_c$ , at least one of the local states  $p_1^i$  or  $p_2^i$  does not belong to  $Q_S^i$ .

The definition of the Global Twin-sWTS is provided in [Definition 21](#).

**Definition 21. Global Twin-sWTS**

Given a Global Twin-WTS

$$\mathcal{V} = (X, X_0, \rightarrow_V, w_V, \mathcal{AP}_g, L_V, A, A_c, Y_g, H_V, Q_{S,g}),$$

its Global Twin-sWTS is a new WTS

$$\mathcal{V}_s = (X_s, X_{0,s}, \rightarrow_s, w_s, \mathcal{AP}_g, L_s, A, A_c, Y_g, H_s, Q_{S,g}),$$

where:

- *The set of states is defined as:*

$$X_s = X \setminus \{(p_1, p_2) = ((p_1^1, \dots, p_1^m), (p_2^1, \dots, p_2^m)) \mid \exists i \in A_c \text{ such that } p_1^i \in Q_S^i \text{ and } p_2^i \in Q_S^i\}$$

- *The set of initial states is defined as:*

$$X_{0,s} = X_0 \setminus \{(p_1, p_2) = ((p_1^1, \dots, p_1^m), (p_2^1, \dots, p_2^m)) \mid \exists i \in A_c \text{ such that } p_1^i \in Q_S^i \text{ and } p_2^i \in Q_S^i\}$$

- $\rightarrow_s \subseteq X_s \times X_s$  is the transition relation defined as: for any  $x_s, x'_s \in X_s$ ,  $(x_s, x'_s) \in \rightarrow_s$  if and only if  $(x_s, x'_s) \in \rightarrow_V$
- $w_s : X_s \times X_s \rightarrow \mathbb{R}_+$  is the cost function inherited by  $\mathcal{V}$  and defined as: for any  $x_s, x'_s \in X_s$  such that  $(x_s, x'_s) \in \rightarrow_s$ ,  $w_s(x_s, x'_s) = w_V(x_s, x'_s)$
- $L_s : X_s \rightarrow \mathcal{AP}_g$  is the labeling function inherited by  $\mathcal{V}$  and defined as: for any  $x_s \in X_s$ ,  $L_s(x_s) = L_V(x_s)$
- $H_s : X_s \rightarrow Y_g$  is the observation function inherited by  $\mathcal{V}$  and defined as: for any  $x_s \in X_s$ ,  $H_s(x_s) = H_V(x_s)$
- $\mathcal{AP}_g, A, A_c, Y_g$  and  $Q_{S,g}$  are directly inherited from  $\mathcal{V}$

### 3.3.2 Type-B Security

Type-B security is a decoy-based approach that relies on agent cooperation to ensure security. An informal overview is provided below to illustrate its core concepts:

**Type-B Security** aims to prevent the intruder from uniquely identifying an agent as the only candidate visitor of a secret state. Specifically, suppose the intruder knows the observations associated with secret states and monitors each agent individually, waiting for such an observation to appear. Once detected, they could track the corresponding agent.

In this setting, the intruder is assumed to know the system's transition relation, secret and initial locations, and observe each agent separately. The goal is to ensure plausible deniability regarding which agent has visited a secret state. To achieve this, if an agent  $i$  visits a secret state, there must exist another agent  $j$  (with  $j \neq i$ ) that simultaneously visits a state with the same observation.

This guarantees that the intruder cannot attribute the secret-state visit to a specific agent with certainty.

### 3.3.2.1 System Model

Type-B security does not require finding copy paths as in Type-A. Instead, only the Global Type-B WTS  $\mathcal{T}_g^B$  is defined, in [Definition 22](#). The global states of  $\mathcal{T}_g^B$  are constructed similarly to those of  $\mathcal{T}_g^A$  (see [Definition 18](#)), with the additional constraint that whenever an agent is in a secret state, another agent must produce the same observation.

#### **Definition 22. Global Type-B WTS**

*By composing the individual WTSs*

$$\mathcal{T}^i = (Q^i, Q_0^i, \rightarrow_i, w_i, \mathcal{AP}_i, L_i, Y_i, H_i, Q_S^i), \forall i \in A$$

*of all  $m$  agents, as defined in [Definition 5](#), a Global Type-B WTS is a new WTS*

$$\mathcal{T}_g^B = \bigotimes_{i=1}^m \mathcal{T}^i = (P, P_0, \rightarrow_g, w_g, \mathcal{AP}_g, L_g, A, Y_g, H_g, Q_{S,g})$$

*where:*

- $P$  is the set of global states  $p = (p^1, p^2, \dots, p^m)$  defined as:

$$P = \left\{ p \in Q^1 \times \dots \times Q^m \mid \forall i \in A, p^i \in Q_S^i \Rightarrow \right. \\ \left. \exists j \neq i \in A \text{ such that } H_i(p^i) = H_j(p^j) \right\}$$

- $P_0$  is the set of initial global states defined as:

$$P = \left\{ p \in Q_0^1 \times \dots \times Q_0^m \mid \forall i \in A, p^i \in Q_S^i \Rightarrow \right. \\ \left. \exists j \neq i \in A \text{ such that } H_i(p^i) = H_j(p^j) \right\}$$

- $\rightarrow_g, w_g, \mathcal{AP}_g, L_g, A, Y_g, H_g, Q_{S,g}$  are defined identically to those in  $\mathcal{T}_g^A$

### 3.3.2.2 Security Property

The security property explained above is formally presented in [Definition 23](#).

#### **Definition 23. Global Path Type-B Security**

A global path  $\tau$  in  $\mathcal{T}_g^B$ , where a local path  $\tau^j = (\tau^j(1), \tau^j(2), \dots, \tau^j(n))$  is defined for each agent  $j \in A$ , is secure if:

- $\forall i = 1, \dots, n$ , and  $\forall j \in A$ : if  $\tau^j(i) \in Q_S^j$ , then there exists at least one other agent  $l \in A$  (with  $l \neq j$ ), such that  $H_j(\tau^j(i)) = H_l(\tau^l(i))$ .

### 3.3.3 Type-C Security

Type-C security is a framework that accounts for potential delays in the intruder's observations. The motivation for this approach is described below:

**Type-C Security** allows an agent's visit to a secret state to be hidden by another agent's visit within a bounded time interval. Specifically, if agent  $i$  visits a secret state at time  $t$ , then there must exist another agent  $j \neq i$  and a time  $t' \in [t, t + k]$  such that  $j$  also visits a secret state.

The motivation behind Type-C security is to hide the identity of the agent visiting a secret state by leveraging the intruder's uncertainty caused by observations with variable delay. Unlike Type-B security, where ambiguity is introduced through identical observations at the same time, Type-C security maintains plausible deniability by allowing a delayed visit to a secret state.

Another key difference between the two is the intruder's ability to observe secret states. In Type-B, the intruder is assumed to infer visits based on an observation function, while in Type-C, the intruder has direct knowledge of when an agent enters a secret state. This makes the previous approach of using copy states ineffective, and an alternative strategy is required to prevent identifying the agent visiting a secret state. However, as in Type-B, the intruder has knowledge of the system's transition relation, secret and initial locations, and observes each agent separately.

Since the intruder seeks to determine which agent has visited a secret state, agents can exploit potential observation delays. By ensuring that another agent visits a secret state within  $k$  time steps, the system preserves uncertainty both the agent and the exact timing of the secret task.

### 3.3.3.1 System Model

Consider a multi-agent system with  $m$  agents, denoted by  $A = \{1, 2, \dots, m\}$ . The motion of each agent  $i \in A$  is modeled by an individual WTS:

$$\mathcal{T}^i = (Q^i, Q_0^i, \rightarrow_i, w_i, \mathcal{AP}_i, L_i, Y_i, H_i, Q_S^i),$$

as defined in [Definition 5](#).

The Type-C Global WTS is constructed by composing the individual WTSs of all  $m$  agents, following a procedure analogous to that used in the Type-A security setting (see [Definition 18](#)). However, since Type-C security does not involve the notion of critical agents, the corresponding subset  $A_c \subseteq A$  is omitted.

Furthermore, it is assumed that the intruder can directly observe whether an agent is in a secret or non-secret state. As a result, the set of observations is defined as  $Y_g = \{\text{secret}, \text{non-secret}\}$ .

The resulting Type-C Global WTS is defined as:

$$\mathcal{T}_g^C = \bigotimes_{i=1}^m \mathcal{T}^i = (P, P_0, \rightarrow_g, w_g, \mathcal{AP}_g, L_g, A, Y_g, H_g, Q_{S,g})$$

### 3.3.3.2 Security Property

The security property explained above is formally presented in [Definition 24](#):

#### **Definition 24. Global Path Type-C Security**

A global path  $\tau$  in  $\mathcal{T}_g^C$ , where a local path  $\tau^j = (\tau^j(1), \tau^j(2), \dots, \tau^j(n))$  is defined for each agent  $j \in A$ , is secure if there exists a constant  $k \geq 0$  such that:

- $\exists i \in \{1, \dots, n\}$ , and  $\exists j \in A$ : if  $\tau^j(i) \in Q_S^j$ , then there exists at least one other agent  $l \in A$  (with  $l \neq j$ ), such that at least one of the states  $\tau^l(i), \tau^l(i+1), \dots, \tau^l(i+k) \in Q_S^l$ .

### 3.3.3.3 Construction of Secure WTS

The framework of Type-C security can handle different intruder capabilities. If the intruder has no delays in perceiving secret state visits, setting  $k = 0$  reduces Type-C security to Type-B security, where another agent  $j$  must visit a secret state at the same time as agent  $i$ . On the other hand, if the intruder's observations are subject to delays of up to  $k$  time steps, setting  $k > 0$  allows for a delayed ambiguity, preserving uncertainty about which agent visited the secret state.

The analysis of Type-C security follows a fundamentally different approach from the previous security types, requiring modifications to the construction of the Global Weighted Transition System (WTS), to account for the  $k$ -step delay window.

Specifically, a set of  $m$  counters  $c^i$ , where  $i \in A$  and  $A$  is the set of  $m$  agents, is introduced. This set is appended to the states of the Global Type-C WTS  $\mathcal{T}_g^C$  forming the states of the Global Type-C Countdown-WTS  $\mathcal{T}_c^C$ . Each counter  $c^i$  takes a value  $v$  from the set  $\{0, 1, \dots, k, \text{None}\}$ . The arithmetic

values  $v$  of the set indicate the remaining time steps within which another agent must visit a secret state. In particular, if agent  $i$  visits a secret state at time step  $t$ , then  $c^i$  is set to  $k$ , meaning that another agent  $j \in A \setminus \{i\}$  must visit a secret state within the next  $k$  time steps. If this condition is met, the counter  $c^i$  is reset to None, signifying that no further visits are needed due to agent  $i$ 's prior secret visit.

The initial global states of  $\mathcal{T}_c^C$  can only have counters  $c^i$ ,  $i \in A$ , set to either None or  $k$ .

In global states where any counter is equal to 1, indicating the last step before violating Type-C security requirements, only transitions that preserve security are allowed. This prevents the system from transitioning to a state where the  $k$ -step delay window is exceeded, thereby preserving Type-C security. For more details, see the definition of the transition relation in [Definition 25](#).

To handle Type-C security, the set of agents  $A$  is partitioned into three disjoint groups,  $G_1$ ,  $G_2$ , and  $G_3$ , such that:

- $G_1 \cup G_2 \cup G_3 = A$ , and
- $G_1 \cap G_2 = G_2 \cap G_3 = G_1 \cap G_3 = \emptyset$ .

Formally:

- An agent  $i \in G_1 \subseteq A$  if its security window is about to expire, i.e.,  $c^i = 1$ .
- An agent  $i \in G_2 \subseteq A$  if it does not have an active countdown, i.e.,  $c^i = \text{None}$ .
- An agent  $i \in G_3 \subseteq A$  if its counter  $c^i$  is active and  $c^i > 1$ .

**Proposition 1.** *There can be at most one agent in  $G_1$ , i.e.,  $|G_1| \leq 1$ .*

*Proof.* Assume that  $|G_1| > 1$ . Then, there exist agents  $A_1, \dots, A_{|G_1|}$  such that for each  $i \in G_1$ ,  $c^i = 1$ , meaning that their countdown expires at the next step.

This implies that all counters  $c^i$  for  $i \in G_1$  were activated exactly  $k - 1$  steps ago, as they started at value  $k$  and decrement by 1 at each step. Therefore, all these agents must have entered secret states  $k - 1$  steps ago.

However, the security condition ensures that if multiple agents enter secret states at the same time, their counters are reset immediately to None.

Hence, none of them should currently have an active counter, contradicting the assumption that  $c^i = 1$  for each  $i \in G_1$ .

Therefore, the assumption must be false, and  $|G_1| \leq 1$ . □

The Global Type-C Countdown-WTS is formally defined as follows:

**Definition 25. Global Type-C Countdown-WTS**

Given a Global Type-C WTS

$$\mathcal{T}_g^C = (P, P_0, \rightarrow_g, w_g, \mathcal{AP}_g, L_g, A, Y_g, H_g, Q_{S,g}),$$

its Global Type-C Countdown-WTS is a new WTS

$$\mathcal{T}_c^C = (P', P'_0, \rightarrow'_g, w'_g, \mathcal{AP}_g, L'_g, A, Y_g, H_g, Q_{S,g}),$$

where:

- $P' \subseteq P \times \{0, 1, \dots, k, \text{None}\}^m$  is the set of states
- $P'_0 = P_0 \times \{k, \text{None}\}^m$  is the set of initial states defined as follows:

For any

$$p = (p_1, p_2, \dots, p_m) \in P_0, \text{ and}$$

$$c = (c_1, c_2, \dots, c_m) \in \{k, \text{None}\}^m,$$

a state  $p' = (p, c) \in P'_0$  if the following conditions hold:

- If the number of agents  $i$  such that  $p_i \in Q_S^i$  is either zero or at least two, then for all  $i \in A$ ,  $c_i = \text{None}$ .
  - If exactly one agent  $i$  satisfies  $p_i \in Q_S^i$ , then  $c_i = k$  for that agent, and  $c_j = \text{None}$  for all  $j \neq i$ .
  - If  $k = 0$ , the number of agents in a secret state must not be exactly one. Otherwise, Type-C security is violated.
- $\rightarrow'_g \subseteq P' \times P'$  is the transition relation, defined as follows:

For any

$$p_1 = (p_1^1, p_1^2, \dots, p_1^m) \in P, \quad p_2 = (p_2^1, p_2^2, \dots, p_2^m) \in P,$$

$$c_1 = (c_1^1, c_1^2, \dots, c_1^m) \in \{0, \dots, k, \text{None}\}^m,$$

$$c_2 = (c_2^1, c_2^2, \dots, c_2^m) \in \{0, \dots, k, \text{None}\}^m,$$

$$p'_1 = (p_1, c_1) \in P', \quad p'_2 = (p_2, c_2) \in P'.$$

the transition  $(p'_1, p'_2) \in \rightarrow'_g$  holds if:

- $(p_1, p_2) \in \rightarrow_g$ , and
- **Case 1: There is one agent in  $G_1$**

Then, at least one agent in  $G_2 \cup G_3$  must visit a secret state and:

- \* If agents from at least two different groups ( $G_1, G_2, G_3$ ) visit a secret state, all counters reset to None.
- \* If exactly one agent  $i \in A$ , from  $G_2 \cup G_3$ , visits a secret state and if  $k > 0$ , then  $c_2^i = k$  and  $c_2^j = \text{None}$ ,  $\forall j \in A \setminus \{i\}$ .
- \* If more than one agent from  $G_2 \cup G_3$  visits a secret state, then  $c_2^j = \text{None}$ ,  $\forall j \in A$ .

The remaining transitions, where:

- \* No agent from  $G_2 \cup G_3$  visits a secret state, or
- \* Exactly one agent from  $G_2 \cup G_3$  visits a secret state and  $k = 0$ , violate Type-C security, thus are invalid.

- **Case 2: There are no agents in  $G_1$ , i.e., all agents are in  $G_2 \cup G_3$**

- \* If exactly one agent  $i \in A$  visits a secret state and if  $k > 0$ , then  $c_2^i = k$  and  $c_2^j = \text{None}$ ,  $\forall j \in A \setminus \{i\}$ .
- \* If more than one agent visits a secret state, then  $c_2^j = \text{None}$ ,  $\forall j \in A$ .
- \* If no agent visits a secret state, then  $c_2^i = c_1^i - 1$ ,  $\forall i \in G_3$ , and  $c_2^i = c_1^i = \text{None}$ ,  $\forall i \in G_2$ .

The remaining transitions, where exactly one agent visits a secret state and  $k > 0$ , violate Type-C security, thus are invalid.

- $w'_g : P' \times P' \rightarrow \mathbb{R}_+$  is the cost function defined as: for any  $p'_1 = (p_1, c_1) \in P'$ ,  $p'_2 = (p_2, c_2) \in P'$  such that  $(p'_1, p'_2) \in \rightarrow'_g$ ,  $w'_g(p'_1, p'_2) = w_g(p_1, p_2)$
- $L'_g : P' \rightarrow \mathcal{AP}_g$  is the labeling function defined as: for any agent  $p' = (p, c) \in P'$ ,  $L'_g(p') = L_g(p)$
- $H'_g : P' \rightarrow Y_g$  is the observation function defined as: for any agent  $p' = (p, c) \in P'$ ,  $H'_g(p') = H_g(p)$
- $\mathcal{AP}_g, A, Y_g$  and  $Q_{S,g}$  are defined identically to those in  $\mathcal{T}_g^C$

### 3.3.4 Motivating Scenarios for Type-A/B/C Security

This thesis introduces three distinct types of security for multi-agent systems, each addressing different intruder’s observational capabilities. To help distinguish them, [Table 3.1](#) summarizes the assumptions for each security type.

The main difference between these security types lies in how the intruder gets information about the agents’ locations:

- In Type-A and Type-B security, the intruder relies on an observation function to infer the agents’ position.
- In Type-C security, the intruder can infer which agent, if any, is at a secret location. However, they may experience a slight delay in receiving this information. This delay creates an opportunity for deception, that is, if multiple agents visit secret locations within a short time window, the intruder will not be able to identify the specific agent responsible for the secret task.

Security Type	Intruder Capabilities
<b>Type-A/B</b>	<ul style="list-style-type: none"> <li>- Observes each agent independently.</li> <li>- Obtains information from an observation function.</li> <li>- Has knowledge of the system’s secret and initial states.</li> <li>- Has knowledge of the system’s transition relation.</li> </ul>
<b>Type-C</b>	<ul style="list-style-type: none"> <li>- Observes each agent independently.</li> <li>- Knows when an agent enters a secret state.</li> <li>- May have a delay in receiving information.</li> <li>- Has knowledge of the system’s secret and initial states.</li> <li>- Has knowledge of the system’s transition relation.</li> </ul>

Table 3.1: Comparison of intruder capabilities for Type-A/B/C security

Each type of security is suited for different real-world scenarios:

### **Type-C Security: Intruder with Knowledge of Secret Locations and Delayed Awareness**

Imagine an intruder monitoring agents in a workplace using surveillance systems installed at secret locations. The cameras inform the intruder whether an agent is in a secret location, but this information may be delayed by up to a pre-defined number of time steps due to transmission or network latency. The intruder also knows when a secret task will be performed, which is modeled as an agent visiting a secret location.

To counteract this, agents can exploit the delay by coordinating their movements. If multiple agents visit secret location within a short period of time, the intruder may not be able to determine which one performed the secret task. By choosing a suspected delay window for the intruder, agents minimize the likelihood of being identified as the one responsible for the secret task.

If the intruder correctly identifies the responsible agent, they can focus their efforts on that agent in the future, compromising the system's security.

### **Type-A Security: Intruder with Indirect Observations**

Consider an intruder who lacks visibility of the agents' location, but gains information about the agents' locations through indirect observations. That is, the intruder infers their positions based on observable clues, such as sensor activations, noise levels, approximate observations by third-party.

Suppose a team of agents where specific subgroups are assigned different roles. The intruder already knows that only a specific subset of agents, referred to as critical agents, are authorized to perform the secret tasks. This knowledge makes them focus on this subgroup of agents.

In this case, in order to guarantee security, the critical agents must always maintain plausible deniability about their locations so that the intruder can never infer with certainty that they visited a secret state.

### **Type-B Security: Intruder with Indirect Observations and Knowledge of Secret State Observations**

Type-B security builds on Type-A but considers an additional risk: even if the intruder relies on indirect observations, the paths taken by the agents may unintentionally expose the agent performing the secret task.

Specifically, if the intruder knows what types of observations correspond to a secret location, and only one agent passes through a state with that

observation, the intruder can immediately identify them as the one performing the secret task. This is a problem when the observation produced by a secret state is unique, or if it happens that no other agent's path passes from a state that produces the same observation.

In this case, Type-B security requires that at least one other agent generates the same observation at the same time as the agent performing the secret task. By doing so, the agents exploit the fact that the intruder relies only on indirect observations, ensuring that the identity of the agent performing the secret task remains ambiguous.

For example applications of these scenarios and their corresponding secure optimal path planning solutions, see [Section 3.5](#).

## 3.4 Secure Optimal Path Planning under LTL Specifications

The optimal path planning algorithm presented in [Algorithm 1](#) can be extended to incorporate security guarantees for both single-agent or multi-agent systems. For multi-agent systems under Type-B or Type-C security, the algorithm is used as-is with  $\mathcal{T}_g^B$  and  $\mathcal{T}_c^C$  as the input WTS, respectively.

For single-agent system security or Type-A multi-agent security, the WTS  $\mathcal{T}$  and the Global WTS  $\mathcal{T}_g^A$  are used as inputs, respectively. However, an additional modification is required to guarantee security. Instead of directly combining the Weighted Transition System (WTS)  $\mathcal{T}$  with the Nondeterministic Büchi Automaton (NBA)  $\mathcal{B}_\phi$ , the Twin-Weighted Transition System (Twin-WTS)  $\mathcal{V}$  is first built from  $\mathcal{T}$ , followed by the Secure Twin-Weighted Transition System (Twin-sWTS)  $\mathcal{V}_s$ . These two function are represented by *buildTwinWTS* and *buildSecureTwinWTS*, respectively, in [Algorithm 2](#).  $\mathcal{V}_s$  is then combined with  $\mathcal{B}_\phi$  to build  $\mathcal{A}_P$ .

Once  $\mathcal{A}_P$  is constructed, the shortest path problem is solved using the same approach as described in [Section 3.1](#). The function returns the optimal twin path  $\tau^*$ , and the secure optimal path is given by  $\Pi(\tau)$ . For completeness, the full algorithm is presented in [Algorithm 2](#), even though the only modification to [Algorithm 1](#) is the addition of lines 1 and 2.

### 3.4.1 Complexity Analysis

In the single-agent case, the WTS consists of  $|\Pi|$  states and the Büchi automaton, derived from the LTL formula, has  $|Q|$  states. Then, the Twin-WTS and the Twin-sWTS are formed by pairing these  $|\Pi|$  states, resulting in at most  $|\Pi|^2$  states. Consequently, the product automaton combines these components, forming a graph with at most  $|\Pi|^2|Q|$  states.

For a multi-agent system, let each individual WTS  $\mathcal{T}^i$ , where  $i \in \{1, 2, \dots, m\}$ , have  $|\Pi_i|$  states. The maximum number of states among all individual WTSs is given by:

$$|\Pi| = \max_{i \in \{1, 2, \dots, m\}} |\Pi_i|.$$

The Büchi automaton has  $|Q|$  states. The product system combines these components to form the graph on which the shortest path problem is solved. The complexity varies depending on the construction of each security type's Global WTS.

---

**Algorithm 2** Secure Optimal Path Planning

---

**Input:** LTL formula  $\phi$ , WTS  $\mathcal{T}$ , weight factor  $\alpha$   
**Output:** optimal twin path  $\tau^*$ , minimum path cost  $J_{\min}$

- 1:  $\mathcal{V} \leftarrow \text{buildTwinWTS}(\mathcal{T})$
- 2:  $\mathcal{V}_s \leftarrow \text{buildSecureTwinWTS}(\mathcal{V})$
- 3:  $\mathcal{B}_\phi \leftarrow \text{buildBüchiAutomaton}(\phi)$
- 4:  $\mathcal{A}_P \leftarrow \text{buildProductBüchiAutomaton}(\mathcal{V}_s, \mathcal{B}_\phi)$
- 5:  $\tau^* \leftarrow \emptyset$
- 6:  $J_{\min} \leftarrow \infty$
- 7: **for** each accepting state  $q_{\text{acc}} \in Q_{\text{acc}}$  **do**
- 8: Compute shortest prefix path  $\pi_{\text{prefix}}$  from any initial state in  $Q_0$  to  $q_{\text{acc}}$
- 9: Compute shortest cycle  $\pi_{\text{cycle}}$  starting from  $q_{\text{acc}}$  and returning to  $q_{\text{acc}}$
- 10: Compute total cost  $J_{\text{total}} = \alpha \cdot J_{\text{prefix}} + (1 - \alpha) \cdot J_{\text{suffix}}$
- 11: **if**  $J_{\text{total}} < J_{\min}$  **then**
- 12:  $J_{\min} \leftarrow J_{\text{total}}$
- 13:  $\tau^* \leftarrow \pi_{\text{prefix}} \oplus \pi_{\text{cycle}}$
- 14: **return**  $\tau^*, J_{\min}$

---

For **Type-A security**, the Global WTS is formed as the product of  $m$  individual WTSs, resulting in at most  $|\Pi|^m$  states. The Global Twin-WTS is constructed by pairing states of the Global WTS, leading to at most  $|\Pi|^{2m}$ . Similarly, the Global Twin-sWTS has at most as many states as the Global Twin-WTS, yielding at most  $|\Pi|^{2m}$  states. The product automaton for Type-A has at most  $|\Pi|^{2m}|Q|$  states. As a result, Type-A security increases the complexity compared to the classic LTL task planning, from at most  $|\Pi|^m|Q|$  to  $|\Pi|^{2m}|Q|$  states. For comparison, [37] solves a similar problem in a graph of at most  $(2|\Pi|)^{2m^2+m}|Q|$  states.

For **Type-B security**, the complexity analysis follows the same reasoning as for Type-A, but without constructing the Twin-WTS and the Twin-sWTS. Consequently, the number of states in the product automaton is at most  $|\Pi|^m|Q|$ , matching the complexity of the classic centralized LTL task planning.

**Type-C security** introduces an important difference in complexity. The states of the Global Type-C Countdown-WTS  $\mathcal{T}_c^C$  are constructed as the product of the  $m$  individual WTS states and  $m$  counters, each associated with a specific agent. Each counter has length of  $(k + 2)$ , leading to at most  $|\Pi|^m(k + 2)^m$  states. The product automaton thus has at most  $|\Pi|^m(k + 2)^m|Q|$  states.

In summary, the dominant factor in complexity arises from the Global

WTS construction, with each security type impacting the state space differently. Type-B maintains the same complexity as centralized LTL planning at  $\mathcal{O}(|\Pi|^m|Q|)$ , Type-A increases it to  $\mathcal{O}(|\Pi|^{2m}|Q|)$ , and Type-C security introduces an additional factor of  $(k + 2)^m$ , resulting in  $\mathcal{O}(|\Pi|^m(k + 2)^m|Q|)$ .

Additionally,  $|Q|$  is bounded by a function of the number of predicates in the LTL formula, making its contribution to overall complexity relatively small.

## 3.5 Motivating Examples

Examples of optimal path planning under LTL specifications and security constraints for both single-agent and multi-agent systems are presented in this Section. In all examples, the constant  $\alpha$ , used to calculate the total path distance, is set to 0.5, assigning equal weight to the path prefix and suffix.

### 3.5.1 Secure Navigation of an Autonomous Robot

In a manufacturing facility, an autonomous robot is responsible for transporting high-value materials. However, certain areas, such as a high-security vault or a confidential research area, must remain hidden from competitors. A malicious intruder tries to determine whether the robot has visited these sensitive locations, hence uncovering vulnerabilities in the factory's operations. However, the spy does not have direct access to the factory's internal tracking systems. Instead, they rely on indirect signals, such as:

- The sound of the robot's motors, which may be heard from multiple locations.
- Motion-triggered lights activating as the robot moves.
- Workers who may notice the robot's presence but not its exact location.

Since multiple locations may produce identical observations, the intruder cannot track the robot's exact path but aims to infer if it has been in a secret location at any point.

The factory floor is modeled as a Weighted Transition System (WTS) (as defined in [Definition 5](#)). States  $Q$  correspond to discrete factory locations:

- A = High-Security Storage Vault (Secret, Starting Point): A restricted storage area containing sensitive materials.
- B = Receiving Dock (Starting Point): A loading/unloading area where materials arrive.
- C = Quality Control Station: An area used for material inspection.
- D = General Production Area: A location where manufacturing processes take place.

- E = Confidential Research Lab (Secret): A restricted area conducting experiments.
- F = Maintenance and Charging Station: A required stop for the robot to recharge and undergo system diagnostics.

The robot can navigate between these locations using the transitions presented in [Figure 3.1](#). Each transition has an associated weight  $w$ , representing the distance the robot needs to cover. Since  $A$  and  $E$  are secret locations, the robot's path must be planned carefully to prevent the intruder from inferring visits to these areas.

The robot's movements generate observations based on an observation function  $H : Q \rightarrow Y$ , where  $Y$  is the set of observable outputs available to the intruder. Since multiple locations may share the same observation, the intruder may not be able to uniquely determine the robot's location. Based on their indirect observations, the intruder concluded that:

$$H(A) = H(B) = H(C) = H(F) = y_1, \quad H(D) = H(E) = y_2.$$

The two observation groups, i.e., states with  $y_1$  and  $y_2$ , are represented in the figure as rectangles and ellipses, respectively.

The states are also assigned with atomic propositions through a labeling function  $L : Q \rightarrow 2^{\mathcal{AP}}$ , where  $\mathcal{AP}$  is the set of atomic propositions. In this setup:

$$L(A) = a, \quad L(B) = b, \quad L(C) = c, \quad L(D) = d, \quad L(E) = e, \quad L(F) = f.$$

The goal is to plan an optimal (shortest) path for the robot while ensuring it completes its required tasks. The robot must satisfy the following Linear Temporal Logic (LTL) formula:

$$\phi = \Box \Diamond e \wedge \Box \Diamond f, \tag{3.2}$$

which translates to:

The robot must visit both the Confidential Research Lab (E) and the  
Maintenance and Charging Station (F) infinitely often.

Additionally, the robot's trajectory must prevent the intruder from determining if it visited the High-Security Vault (A) or the Research Lab (E).

The simple Weighted Transition System (WTS) modeling this example is illustrated in [Figure 3.1](#). The initial states, marked in red, are given by  $Q_0 =$

$\{A, B\}$ . The secret states, highlighted in light blue, are  $Q_S = \{A, E\}$ . State  $A$  appears in both, so it is shown with both colors. The Twin-WTS is constructed

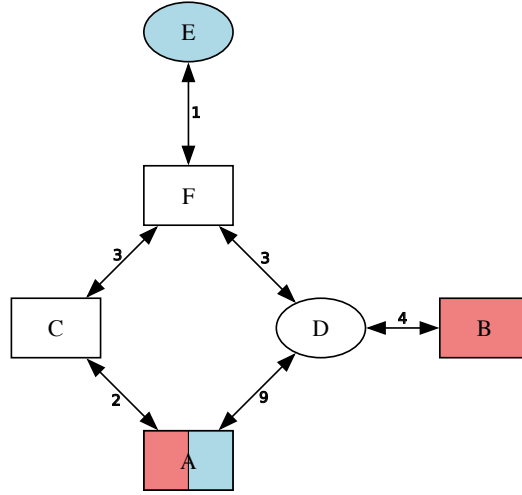


Figure 3.1: An example WTS with initial states in red and secret states in light blue)

by pairing states of the original WTS that share the same observation, as described above. The Twin-sWTS is then obtained by removing from the Twin-WTS all states of the form  $(q_1, q_2) \in X$  where both  $q_1$  and  $q_2$  belong to  $Q_S$ . These two WTS are illustrated in Figure 3.2 and Figure 3.3, where transition weights are omitted for simplicity. The initial states are still depicted by red.

By comparing the two transition systems, it is observed that the states present in the Twin-WTS but absent in the Twin-sWTS are  $(A, A)$  and  $(E, E)$ , as the set of secret states is  $Q_S = \{A, E\}$ . Therefore, the projection of every path in the Twin-sWTS onto the original WTS is secure-by-construction.

Running the secure optimal path planning algorithm with the LTL formula from Equation 3.2 yields the optimal path

$$B \rightarrow D \rightarrow [F \rightarrow E]^\omega$$

with a total cost of 4.5. The corresponding copy path is:

$$A \rightarrow D \rightarrow [A \rightarrow D]^\omega$$

The real path includes the Confidential Research Lab  $E$ , a restricted area. To maintain security, the copy path substitutes  $E$  with the General Production

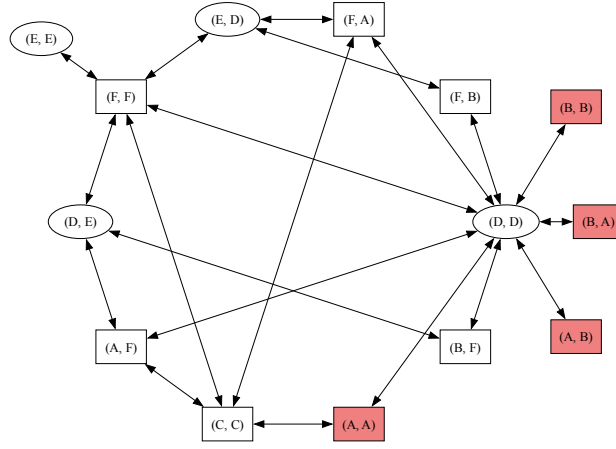


Figure 3.2: The Twin-WTS corresponding to the WTS in [Figure 3.1](#)

Area  $D$ , an unrestricted location that shares the same observation as  $E$ , ensuring the paths remain indistinguishable to the intruder.

### 3.5.2 Secure Navigation of a Multi-Agent Robot Team

#### 3.5.2.1 Type-A Security

Building on the single-agent example, a multi-agent scenario is now considered where two autonomous robots collaborate to transport high-value materials while ensuring security. The robots navigate the same WTS shown in [Figure 3.1](#), and their joint behavior is captured by a Global WTS. This is constructed by taking the Cartesian product of two identical WTSs. Each state in the Global WTS represents a pair of locations, indicating where both robots are at a given time. The Global WTS is presented in [Figure 3.4](#). The secret locations remain the same as in the single-agent case:  $A$  (High-Security Storage Vault) and  $E$  (Confidential Research Lab).

The robots must now satisfy the following LTL formula:

$$\phi = (\diamond c^1 \vee \diamond c^2) \wedge \square \diamond e^1 \wedge \square \diamond f^1 \wedge \square \diamond f^2,$$

which translates to:

- At least one robot must visit the Quality Control Station  $C$  at some point.
- Robot 1 must visit the Confidential Research Lab  $E$  infinitely often.

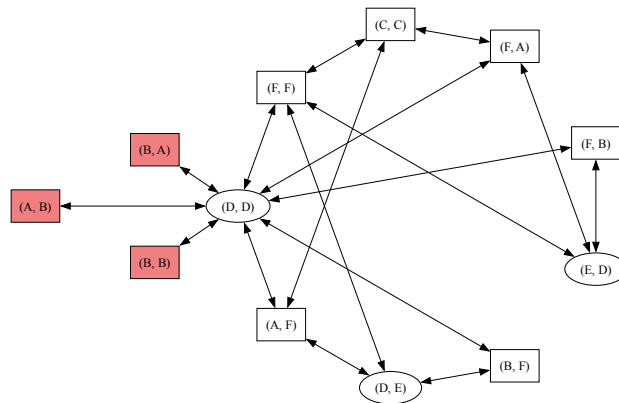


Figure 3.3: The Twin-sWTS corresponding to the WTS in [Figure 3.1](#)

- Both robots must visit the Maintenance and Recharge Station ( $F$ ) infinitely often.

As in the single-agent case, the Twin-WTS is constructed by pairing states with identical observations, that is, states that are indistinguishable from the intruder's point of view. The Twin-sWTS removes pairs where either robot is in a secret state in both the real and copy states, constructing in this way a secure state space. Due to the large state space, the Global Twin-WTS and the Global Secure Twin-WTS are not depicted. To compare different security requirements, four scenarios are evaluated:

- Only Robot 1 is a critical agent, so its visits to secret locations must remain hidden.
- Only Robot 2 is a critical agent, so the same conditions apply, but for Robot 2.
- Both agents are critical, requiring both to avoid revealing visits to secret locations.
- No agent is critical, so both robots can freely visit secret states and the problem reduces to standard multi-agent path planning.

The optimal prefix-suffix paths computed for each scenario, along with their respective total distances, are summarized in [Table 3.2](#).

The secret states  $A$  and  $E$  must remain hidden for critical agents, so the copy path replaces them with observationally equivalent states to ensure

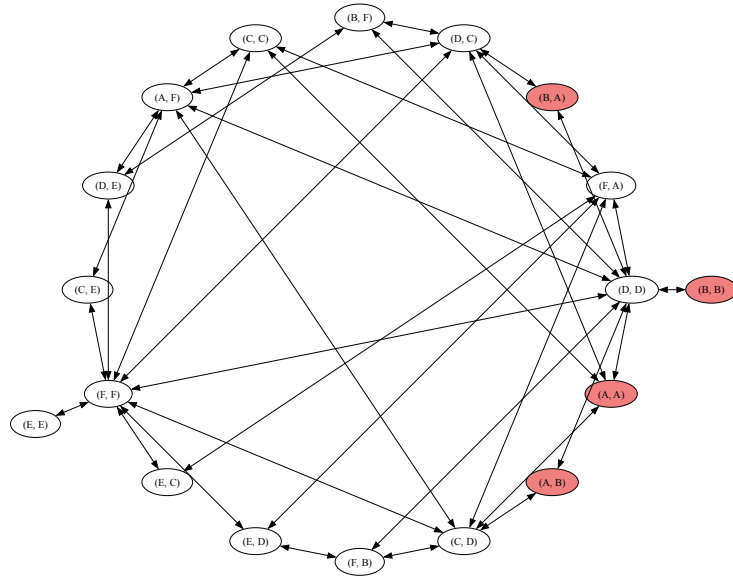


Figure 3.4: The Global WTS constructed from two WTSs, both identical to the one in [Figure 3.1](#)

security. The longest optimal path distance occurs when both agents are critical, as this scenario imposes the strictest constraints on their movement. Conversely, when no agent is critical, the problem reduces to standard LTL-based multi-agent path planning, resulting to the shortest path. In this case, security is irrelevant, making the optimal and copy paths identical.

### 3.5.2.2 Type-B Security

In Type-B security, the goal is to confuse the intruder about what agent performed a secret task. This is achieved by requiring that whenever an agent visits a secret state, another agent must simultaneously be in a state that produces the same observation.

To illustrate this, two agents once again navigate the WTS of [Figure 3.1](#) with minor changes. This time, Robot 1 always starts from state  $A$  and Robot 2 starts from state  $B$ . The secret state is  $C$ . All other transitions and observations remain unchanged. The same LTL formula is used:

$$\phi = (\diamond c^1 \vee \diamond c^2) \wedge \square \diamond e^1 \wedge \square \diamond f^1 \wedge \square \diamond f^2$$

Running the secure optimal path planning algorithm for Type-B security

Table 3.2: Optimal and copy paths with shortest distances for different critical agent assignments and Type-A security

Critical Agent	Global Paths	Cost
Robot 1	<p><b>Optimal Path:</b>  <math>(B, A) \rightarrow (D, C) \rightarrow [(F, F) \rightarrow (E, E)]^\omega</math></p> <p><b>Copy Path:</b>  <math>(A, A) \rightarrow (D, C) \rightarrow [(A, F) \rightarrow (D, E)]^\omega</math></p>	8.0
Robot 2	<p><b>Optimal Path:</b>  <math>(A, B) \rightarrow (C, D) \rightarrow [(F, F) \rightarrow (E, E)]^\omega</math></p> <p><b>Copy Path:</b>  <math>(A, B) \rightarrow (C, D) \rightarrow [(F, A) \rightarrow (E, D)]^\omega</math></p>	8.0
Both robots	<p><b>Optimal Path:</b>  <math>(B, B) \rightarrow (D, D) \rightarrow (F, F) \rightarrow (C, E) \rightarrow [(F, F) \rightarrow (E, E)]^\omega</math></p> <p><b>Copy Path:</b>  <math>(B, B) \rightarrow (D, D) \rightarrow (A, A) \rightarrow (C, D) \rightarrow [(A, A) \rightarrow (D, D)]^\omega</math></p>	13.0
No robot	<p><b>Optimal Path:</b>  <math>(A, A) \rightarrow (C, C) \rightarrow [(F, F) \rightarrow (E, E)]^\omega</math></p> <p><b>Copy Path:</b>  <math>(A, A) \rightarrow (C, C) \rightarrow [(F, F) \rightarrow (E, E)]^\omega</math></p>	7.0

yields the optimal path

$$(A, B) \rightarrow (D, D) \rightarrow (F, F) \rightarrow (C, C) \rightarrow [(F, F) \rightarrow (E, E)]^\omega$$

with a total cost of 17.5. In contrast, the corresponding optimal path without security constraints is:

$$(A, B) \rightarrow (C, D) \rightarrow [(F, F) \rightarrow (E, E)]^\omega$$

This path has a significantly smaller distance of 8.0. The increased cost in the secure path is justified since Robot 1 cannot immediately visit  $C$  in the first step, as it is a secret state, and Robot 2 is unable to produce the same observation simultaneously. Consequently, Robot 1 must first transition to  $D$  despite its high transition cost. By the third time step, both agents visit  $C$  together, ensuring that the intruder cannot determine which agent performed the secret task.

The resulting Type-B secure optimal is shown in the left subfigure of [Figure 3.5](#), along with the optimal path without security guarantees in the right

subfigure.

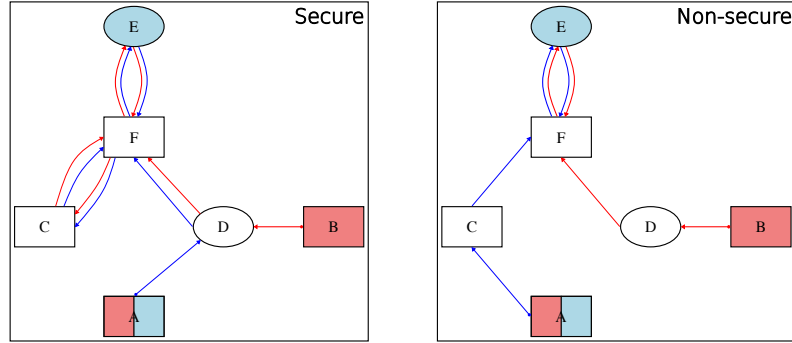


Figure 3.5: Optimal paths with and without Type-B security guarantees

### 3.5.2.3 Type-C Security

In Type-C security, the intruder can identify what agent visits secret states. However, there may be a slight delay in their observations, which the agents can exploit in order to hide the identity of the agent performing a secret task. This is accomplished by introducing a  $k$ -step window, requiring that within  $k$  steps of an agent visiting a secret state, another agent must also visit a secret state.

As an example, two agents navigate the WTS of [Figure 3.1](#) with minor modifications. Robot 1 always starts from state  $A$  and Robot 2 starts from state  $B$ . All transitions and observations remain unchanged, and the same LTL formula is used.

For  $k = 0$  or  $k = 1$ , meaning that another agent must visit a secret state in the same or the immediate next step after an agent visits one, no feasible solution exists. This is because state  $A$  is a secret state and the starting position of Robot 1. Meanwhile, Robot 2 starts from  $B$  and cannot reach a secret state within one transition.

For  $k = 2$ , the secure optimal path planning algorithm leads to the following optimal path:

$$(A, B) \rightarrow (C, D) \rightarrow (F, A) \rightarrow (E, C) \rightarrow [(F, F) \rightarrow (E, E)]^\omega,$$

with a total cost of 14.5.

For any  $k > 2$ , the optimal Type-C secure path coincides with the path obtained without considering security:

$$(A, B) \rightarrow (C, D) \rightarrow [(F, F) \rightarrow (E, E)]^\omega,$$

with a total cost of 8.0.

This occurs because, under this LTL task, setting  $k > 2$  does not introduce any additional constraints on the system. The optimal path happens to satisfy Type-C security for any  $k > 2$ . This means that if the intruder is assumed to have a delay of more than 2 time steps, their presence does not impact the system's behavior under this task.

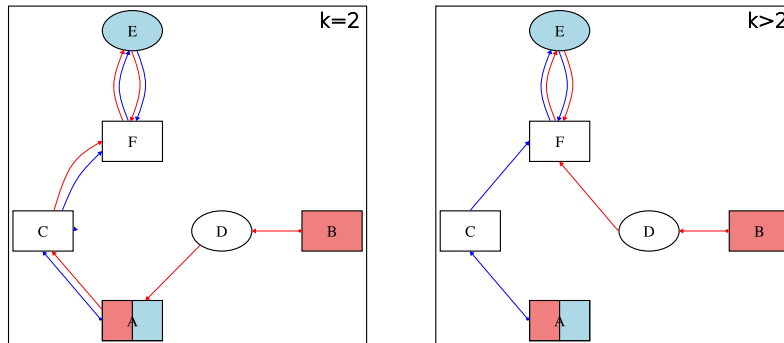


Figure 3.6: Type-C secure optimal paths for all  $k \geq 2$

## Chapter 4

# Secure-by-Construction Trajectory Planning for Continuous Linear Systems with LTL Specifications

This Chapter extends the secure-by-construction path planning framework under Linear Temporal Logic (LTL) specifications to continuous systems. In this context, continuous systems refer to single-agent or multi-agent robotic systems that operate in continuous  $N$ -dimensional workspaces according to affine dynamics.

## 4.1 Single-Agent Systems

### 4.1.1 Problem Setting

Unlike the discrete setting in [Chapter 3](#), this Chapter introduces two key extensions: first, agents are no longer restricted to discrete state spaces but can move in continuous  $N$ -dimensional workspaces; second, their motion is governed by continuous dynamics, allowing for a more realistic modeling of robotic behavior.

#### 4.1.1.1 System Model

The agents' dynamics are given by:

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} + \mathbf{b},$$

where  $\mathbf{x} \in \mathbb{R}^N$  represents the position vector,  $\mathbf{u} \in U \subset \mathbb{R}^m$  is the input vector,  $\mathbf{A} \in \mathbb{R}^{N \times N}$  and  $\mathbf{B} \in \mathbb{R}^{N \times m}$  are constant matrices, and  $\mathbf{b} \in \mathbb{R}^N$  is a constant vector. Here,  $U \subset \mathbb{R}^m$  denotes the input set, indicating that all control inputs are bounded.

The workspace  $P \subset \mathbb{R}^N$  is defined as a full-dimensional polytope consisting of all points  $\mathbf{x} \in \mathbb{R}^N$  that satisfy a finite set of linear inequalities:

$$P = \{\mathbf{x} \in \mathbb{R}^N \mid \mathbf{A}_P \mathbf{x} + \mathbf{b}_P \leq \mathbf{0}\}. \quad (4.1)$$

In other words,  $P$  is the intersection of a finite number of closed half-spaces in  $\mathbb{R}^N$ , resulting in a convex, bounded set with non-empty interior.

#### 4.1.1.2 Security Property

To define security for continuous multi-agent systems, it is first needed to extend the notion of external paths (see [Definition 8](#)) to the continuous domain through the concept of external trajectories. Accordingly, the observation function  $H$  is redefined to act on continuous trajectories  $\mathbf{x}(t)$ , producing an external trajectory  $\mathbf{p}(t) = H(\mathbf{x}(t))$ .

#### **Definition 26. External Trajectory**

*Given a trajectory  $\mathbf{x}(t)$ ,  $t \in [0, t_f]$ , in  $P$ , its external trajectory is  $\mathbf{p}(t) = H(\mathbf{x}(t))$ ,  $t \in [0, t_f]$ .*

The security property for continuous systems can be formally states as follows:

**Definition 27. Continuous Trajectory Security**

A continuous trajectory  $\mathbf{x}(t)$  is secure if there exists a trajectory  $\mathbf{x}'(t)$  in  $P$  such that:

- The two trajectories share the same external trajectory, i.e.,  $H(\mathbf{x}(t)) = H(\mathbf{x}'(t)), \forall t \in [0, t_f]$ , and
- If  $\mathbf{x}$  is in a secret state during a time interval  $[t_1, t_2]$ , where  $t_2 > t_1 \geq 0$ , i.e.,  $\mathbf{x}(t) \in Q_S, \forall t \in [t_1, t_2]$ , then  $\mathbf{x}'$  is not in a secret state, i.e.,  $\mathbf{x}'(t) \notin Q_S, \forall t \in [t_1, t_2]$ .

Intuitively, this definition mirrors the discrete case: a trajectory is secure if there exists an indistinguishable copy trajectory (from the intruder’s perspective) that avoids entering secret regions whenever the true trajectory does.

### 4.1.2 Workspace Partitioning

The predicates  $\pi_i$ , for  $i = 1, 2, \dots, n_\pi$ , that appear in the LTL formula belong to the set of atomic propositions  $\mathcal{AP}$ , defined as:

$$\mathcal{AP} = \{\pi_i \mid i = 1, 2, \dots, n_\pi\}.$$

Each predicate  $\pi_i$  is associated with a region in the state space where it evaluates to true. Specifically, this region is defined as the intersection of the workspace  $P \subset \mathbb{R}^N$  and a union of  $l_i$  open half-spaces, which together form a sub-polytope where  $\pi_i$  holds. The corresponding sub-polytope  $P_i \subset \mathbb{R}^N$  is defined as follows:

$$P_i = \{\mathbf{x} \in \mathbb{R}^N \mid \mathbf{A}_P \mathbf{x} + \mathbf{b}_P \leq \mathbf{0} \text{ and } \mathbf{A}_i \mathbf{x} + \mathbf{b}_i < \mathbf{0}\},$$

where  $\mathbf{A}_i \in \mathbb{R}^{l_i \times N}$  and  $\mathbf{b}_i \in \mathbb{R}^{l_i}$  define the strict linear inequalities corresponding to the  $l_i$  open half-spaces.

In other words, a point  $\mathbf{x} \in \mathbb{R}^N$  satisfies predicate  $\pi_i$  if and only if it lies within both the workspace polytope  $P$  and the region defined by the strict inequalities, i.e.,  $\mathbf{x} \in P_i$ .

This setup follows the framework developed by Kloetzer and Belta [16]. An extension introduced in this thesis is the incorporation of additional regions of interest, defined analogously to predicate regions. These include:

- Initial regions, representing robot's potential starting locations,
- Secret regions, which correspond to the continuous counterpart of secret discrete states, and
- Observation-equivalence regions, where all points within the same region are indistinguishable from the intruder's perspective.

These regions are introduced to address security-oriented specifications and facilitate the application of the planning algorithms presented in [Chapter 3](#). Each secret region is defined as follows:

$$P_{S,i} = \{\mathbf{x} \in P \mid \mathbf{A}_P \mathbf{x} + \mathbf{b}_P \leq \mathbf{0} \text{ and } \mathbf{C}_i \mathbf{x} + \mathbf{d}_i < \mathbf{0}\}, i = 1, 2, \dots, n_S$$

where  $\mathbf{C}_i \in \mathbb{R}^{l_i \times N}$ ,  $\mathbf{d}_i \in \mathbb{R}^{l_i}$ .

The observation-equivalence regions are similarly defined as:

$$P_{\text{obs},i} = \{\mathbf{x} \in P \mid \mathbf{A}_P \mathbf{x} + \mathbf{b}_P \leq \mathbf{0} \text{ and } \mathbf{E}_i \mathbf{x} + \mathbf{f}_i < \mathbf{0}\}, i = 1, 2, \dots, n_{\text{obs}} - 1$$

where  $\mathbf{E}_i \in \mathbb{R}^{l_i \times N}$ ,  $\mathbf{f}_i \in \mathbb{R}^{l_i}$ . Each region  $P_{\text{obs},i}$  contains all points in the workspace  $P$  that yield an identical observation from the perspective of the intruder, thus representing indistinguishability under the chosen observation function  $H$ .

The final observation-equivalence region is defined as the complement of the union of the previous ones:

$$P_{\text{obs},n_{\text{obs}}} = \left\{ \mathbf{x} \in P \mid \mathbf{x} \notin \bigcup_{i=1}^{n_{\text{obs}}-1} P_{\text{obs},i} \right\},$$

and includes all remaining points in the workspace not covered by any of the previously defined observation-equivalence regions.

The initial regions are defined as:

$$P_{\text{init},i} = \{\mathbf{x} \in P \mid \mathbf{A}_P \mathbf{x} + \mathbf{b}_P \leq \mathbf{0} \text{ and } \mathbf{G}_i \mathbf{x} + \mathbf{h}_i < \mathbf{0}\}, i = 1, 2, \dots, n_{\text{init}}$$

where  $\mathbf{G}_i \in \mathbb{R}^{l_i \times N}$ ,  $\mathbf{h}_i \in \mathbb{R}^{l_i}$ .

The final partitioning of the workspace  $P$  is based on atomic predicates, initial regions, secret regions, and observation equivalence. Consequently, at

least one of the following aspects will differ between any two sub-polytopes: initial status (whether it is an initial region or not), secret status (whether it is a secret region or not), the predicates that hold, or the observation from the intruder's perspective.

### 4.1.3 Centroid-Based Transitions

Now that the continuous workspace has been discretized, it can be abstracted as a Weighted Transition System (WTS). However, a key challenge arises: defining the transition relation, that is, determining which transitions are feasible and under which control inputs.

To address this, an important assumption is made: agents are restricted to move only between characteristic points in the workspace. These points are chosen as the centroids (geometric centers) of the sub-polytopes resulting from the workspace partitioning. This assumption is necessary to ensure LTL task satisfaction and security in continuous systems.

Allowing transitions between arbitrary points in the workspace introduces significant complications. In both single-agent and multi-agent systems, discrete paths, represented by sequences of twin states for single agents or global states for multiple agents, must evolve simultaneously to guarantee LTL task satisfaction and security.

If transitions were allowed between arbitrary points, the time required to move from one state to another would depend on the endpoint of the previous transition. This variability makes transitions path-dependent and prevents synchronization across transitions of all agents. As a result, it would be impossible to reliably translate a discrete plan into a continuous trajectory that guarantees both LTL satisfaction and security. Instead, a trial-and-error approach would be needed, repeatedly verifying if discrete plans result in secure, task-compliant trajectories. Such an approach is inefficient, lacks formal guarantees, and contradicts the core objective of this thesis: to achieve secure-by-construction trajectory planning.

It is important to note that this assumption makes the continuous trajectory planning problem solution incomplete. That is, there may exist feasible trajectories that satisfy the LTL task and preserve security as defined in [Definition 27](#), without passing through centroids. On the other hand, the correctness of the proposed solution is later discussed.

#### 4.1.3.1 Construction of the Transition Relation

The goal is to represent the discretized workspace as a WTS

$$\mathcal{T} = (Q, Q_0, \rightarrow, w, \mathcal{AP}, L, Y, H, Q_S),$$

as defined in **Definition 5**. Among its components, the transition relation  $\rightarrow$  remains to be explicitly defined. Specifically, it must be determined which transitions between sub-polytopes  $q, q' \in Q$  are feasible under the system's dynamics.

As an initial step, a set of candidate transitions is established:

- Self-transitions:  $(q, q')$  such that  $q = q'$ .
- Transitions between adjacent sub-polytopes:  $(q, q')$  such that  $q$  and  $q'$  share a common facet.

Following the methodology introduced in the discrete framework, a Twin-Weighted Transition System (Twin-WTS) is constructed:

$$\mathcal{V} = (X, X_0, \rightarrow_V, w_V, \mathcal{AP}, L_V, Y, H_V, Q_S),$$

as defined in **Definition 15**, and a Secure Twin-WTS (Twin-sWTS)

$$\mathcal{V}_s = (X_s, X_{0,s}, \rightarrow_s, w_s, \mathcal{AP}, L_s, Y, H_s, Q_S),$$

as defined in **Definition 16**. Each transition in the Twin-sWTS is of the form  $((q_1, q_2), (q_3, q_4)) \in \rightarrow_s$ , where  $(q_1, q_3)$  is the real transition and  $(q_2, q_4)$  is the corresponding copy transition, both taken from the candidate set. Let  $\mathbf{x}_{c,q_i}$  denote the centroid of the sub-polytope  $q_i$ ,  $i = 1, \dots, 4$ .

To validate a Twin-sWTS transition, it is necessary to confirm that both the real and copy transitions are feasible under the system dynamics and all imposed constraints:

- **System dynamics:** The trajectories of both the real and copy agents must follow continuous-time affine dynamics:

$$\begin{aligned} \dot{\mathbf{x}}_1(t) &= \mathbf{A}\mathbf{x}_1(t) + \mathbf{B}\mathbf{u}_1(t) + \mathbf{b}, & t \in [0, t_f], \\ \dot{\mathbf{x}}_2(t) &= \mathbf{A}\mathbf{x}_2(t) + \mathbf{B}\mathbf{u}_2(t) + \mathbf{b}, & t \in [0, t_f], \end{aligned}$$

where the subscripts 1 and 2 refer to the real and the copy agent, respectively.

- **Initial and final conditions:** The trajectories must start and end at the centroids of the source and target sub-polytopes simultaneously:

$$\begin{aligned}\mathbf{x}_1(0) &= \mathbf{x}_{c,q_1}, & \mathbf{x}_1(t_f) &= \mathbf{x}_{c,q_3} \\ \mathbf{x}_2(0) &= \mathbf{x}_{c,q_2}, & \mathbf{x}_2(t_f) &= \mathbf{x}_{c,q_4}\end{aligned}$$

- **Control constraints:** The inputs must remain within the admissible control set at all times:

$$\mathbf{u}_1(t), \mathbf{u}_2(t) \in U, \quad t \in [0, t_f]$$

- **Workspace constraints:** The trajectories must remain within the workspace throughout the transition:

$$\begin{aligned}\mathbf{A}_P \mathbf{x}_1(t) + \mathbf{b}_P &\leq \mathbf{0}, & t &\in [0, t_f], \\ \mathbf{A}_P \mathbf{x}_2(t) + \mathbf{b}_P &\leq \mathbf{0}, & t &\in [0, t_f]\end{aligned}$$

- **Direct transitions:** The trajectories must remain strictly within the union of their source and target sub-polytopes:

$$\begin{aligned}\mathbf{H}_1 \mathbf{x}_1(t) + \mathbf{g}_1 &< \mathbf{0}, & t &\in [0, t_f], \\ \mathbf{H}_2 \mathbf{x}_2(t) + \mathbf{g}_2 &< \mathbf{0}, & t &\in [0, t_f],\end{aligned}$$

where these inequalities ensure that  $\mathbf{x}_1(t) \in q_1 \cup q_3$  and  $\mathbf{x}_2(t) \in q_2 \cup q_4$ , at all times.

- **Simultaneous facet crossing:** To ensure synchronized transitions between adjacent polytopes, both the real and the copy agent must cross their respective shared facet at the same time:

$$\begin{aligned}\mathbf{a}_1^T \mathbf{x}_1(t_{cross}) + b_1 &= 0, \\ \mathbf{a}_2^T \mathbf{x}_2(t_{cross}) + b_2 &= 0,\end{aligned}$$

where these equalities define the hyperplanes corresponding to the shared facets between adjacent sub-polytopes.

For a transition from the centroid of state  $p_1$  to the centroid of  $p'_1$ , two time durations are critical:

- $t_{cross}$ : the time required to reach the shared facet of  $p_1$  and  $p'_1$ , starting from the centroid of  $p_1$ , and

- $t_f - t_{cross}$ : the time required to move from the shared facet to the the centroid of  $p'_1$ .

The sum of these durations represents the total time  $t_f$  required for the transition between the centroids of  $p_1$  and  $p'_1$ .

For twin transitions where either the real or the copy agent performs a self-transition, the simultaneous facet-crossing constraint is omitted.

Given the constraints above, the problem of determining transition feasibility can be naturally formulated as an optimization problem.

The objective is to minimize execution time. The resulting optimization problem is:

**Problem 1. Continuous-Time Trajectory Optimization**

$$\begin{aligned}
 \min \quad & t_f \\
 \text{s.t.} \quad & \dot{\mathbf{x}}_j(t) = \mathbf{A}\mathbf{x}_j(t) + \mathbf{B}\mathbf{u}_j(t) + \mathbf{b}, \quad t \in [0, t_f], \quad t_f \in \mathbb{R}_+, \quad j = 1, 2 \\
 & \mathbf{x}_j(0) = \mathbf{x}_{c,p_1^j}, \quad \mathbf{x}_j(t_f) = \mathbf{x}_{c,p_2^j}, \quad j = 1, 2 \\
 & \mathbf{u}_j(t) \in U, \quad t \in [0, t_f], \quad j = 1, 2 \\
 & \mathbf{A}_P\mathbf{x}_j(t) + \mathbf{b}_P \leq \mathbf{0}, \quad t \in [0, t_f], \quad j = 1, 2 \\
 & \mathbf{H}_j\mathbf{x}_j(t) + \mathbf{g}_j < \mathbf{0}, \quad t \in [0, t_f], \quad j = 1, 2 \\
 & \mathbf{a}_j^T \mathbf{x}_j(t_{cross}) + b_j = 0, \quad j = 1, 2
 \end{aligned}$$

To solve this problem numerically, the time interval  $[0, t_f]$  is discretized into  $n$  steps. The first  $n_{cross}$  steps have size  $dt_1$  while the remaining  $n_{after}$  have size  $dt_2$ , such that

$$t_{cross} = n_{cross} \cdot dt_1, \text{ and } t_f = t_{cross} + n_{after} \cdot dt_2.$$

The dynamics are approximated using the forward Euler method:

$$\begin{aligned}
 \mathbf{x}_j(i+1) &= \mathbf{x}_j(i) + dt_1 \cdot (\mathbf{A}_j\mathbf{x}_j(i) + \mathbf{B}_j\mathbf{u}_j(i) + \mathbf{b}_j), \\
 & \quad i = 0, \dots, n_{cross} - 1, \quad j = 1, 2 \\
 \mathbf{x}_j(i+1) &= \mathbf{x}_j(i) + dt_2 \cdot (\mathbf{A}_j\mathbf{x}_j(i) + \mathbf{B}_j\mathbf{u}_j(i) + \mathbf{b}_j), \\
 & \quad i = n_{cross}, \dots, n - 1, \quad j = 1, 2
 \end{aligned}$$

The discrete-time optimization problem becomes:

**Problem 2. Discrete-Time Trajectory Optimization**

$$\min \quad t_f$$

$$s.t. \quad t_{cross} = n_{cross} \cdot dt_1, \quad dt_1 > 0$$

$$t_f = t_{cross} + n_{after} \cdot dt_2, \quad dt_2 > 0$$

$$\mathbf{x}_j(i+1) = \mathbf{x}_j(i) + dt_1 \cdot (\mathbf{A}\mathbf{x}_j(i) + \mathbf{B}\mathbf{u}_j(i) + \mathbf{b}),$$

$$i = 0, \dots, n_{cross} - 1, \quad j = 1, 2$$

$$\mathbf{x}_j(i+1) = \mathbf{x}_j(i) + dt_2 \cdot (\mathbf{A}\mathbf{x}_j(i) + \mathbf{B}\mathbf{u}_j(i) + \mathbf{b}),$$

$$i = n_{cross}, \dots, n - 1, \quad j = 1, 2$$

$$\mathbf{x}_j(0) = \mathbf{x}_{c,p_1^j}, \quad \mathbf{x}_j(n) = \mathbf{x}_{c,p_2^j}, \quad j = 1, 2$$

$$\mathbf{u}_j(i) \in U, \quad i = 0, \dots, n - 1, \quad j = 1, 2$$

$$\mathbf{A}_P \mathbf{x}_j(i) + \mathbf{b}_P \leq \mathbf{0}, \quad i = 0, \dots, n, \quad j = 1, 2$$

$$\mathbf{H}_j \mathbf{x}_j(i) + \mathbf{g}_j < \mathbf{0}, \quad i = 0, \dots, n, \quad j = 1, 2$$

$$\mathbf{a}_j^T \mathbf{x}_j(n_{cross}) + b_j = 0, \quad j = 1, 2$$

Note that  $dt_1$  and  $dt_2$  are also decision variables of the problem, making **Problem 2** nonlinear due to their presence in the discrete-time dynamics. The total number of decision variables is

$$2 + 2 \cdot (n + 1) \cdot N + 2 \cdot n \cdot m,$$

corresponding to:

- the two time steps,  $dt_1$  and  $dt_2$ ,
- $n + 1$  discrete positions  $\mathbf{x}(i)$  (in  $P \subset \mathbb{R}^N$ ) for each agent, and
- $n$  control inputs  $\mathbf{u}(i)$  (in  $\mathbb{R}^m$ ) for each agent.

In conclusion, a discrete-time trajectory optimization problem is solved for each candidate transition in the Twin-sWTS. If a solution is found, the corresponding transition is kept in the Twin-sWTS, and the resulting duration  $t_f$  is assigned as its transition weight. Otherwise, the transition is discarded.

#### 4.1.4 Secure Trajectory Planning under LTL Specifications

After constructing the transition relation of the Twin-sWTS, the secure path planning procedure described in [Algorithm 2](#) is applied. Given an LTL formula, the algorithm generates a secure discrete path in prefix-suffix form.

A controller is guaranteed to exist for executing the discrete plan, as the feasibility of each transition (between the centroids of any pair of sub-polytopes) has already been verified during the construction of the Twin-sWTS transition relation.

Thus, if a discrete path is found, the corresponding real and copy trajectories can be computed by concatenating the trajectory segments obtained from solving [Problem 2](#) for each transition. These trajectories are guaranteed to satisfy trajectory security as defined in [Definition 27](#), since they execute all the discrete transitions in the path in the correct order and simultaneously. As a result, LTL specifications and security properties are fulfilled.

Regarding completeness, the method is not complete due to three main limitations. First, it restricts the agent to move only between sub-polytope centroids, which may exclude feasible paths. Second, discretizing the trajectory introduces approximation errors, as the control input is assumed to remain constant over each time step. Third, the requirement that the real and copy trajectories cross simultaneously may not be needed for all transitions. In some cases, non-simultaneous crossings may still preserve security. This constraint could be relaxed by individually verifying whether a particular transition requires simultaneous crossings. However, selectively removing it in multi-agent systems can compromise the satisfaction of LTL specifications.

For instance, consider the transition  $((q_1, q_2), (q_3, q_3))$ . If the LTL formula requires that the two agents visit  $q_3$  simultaneously and prohibits them from occupying  $q_3$  separately, then a non-simultaneous crossing would violate the LTL specification, even though the discrete plan itself is valid.

## 4.2 Multi-Agent Systems

The framework introduced earlier in this Chapter can be naturally extended to support Type-A and Type-B security for multi-agent systems, with minor modifications to the single-agent secure trajectory planning approach.

In contrast, supporting Type-C security requires a different construction of the Global Type-C Countdown-WTS than the one presented in [Definition 25](#).

This Section formally defines trajectory security for all three security types and explains how secure trajectory planning is implemented for each.

### 4.2.1 System Model

The system consists of  $m$  agents, each governed by affine dynamics:

$$\dot{\mathbf{x}}^i = \mathbf{A}^i \mathbf{x}^i + \mathbf{B}^i \mathbf{u}^i + \mathbf{b}^i, \quad i = 1, \dots, m$$

where  $\mathbf{x}^i \in \mathbb{R}^N$  is the position vector of agent  $i$ ,  $\mathbf{u} \in U^i \subset \mathbb{R}^m$  is the input vector,  $\mathbf{A}^i \in \mathbb{R}^{N \times N}$  and  $\mathbf{B}^i \in \mathbb{R}^{N \times m}$  are constant matrices, and  $\mathbf{b}^i \in \mathbb{R}^N$  is a constant vector. The input constraint sets  $U^i \subset \mathbb{R}^m, \forall i = 1, \dots, m$ , bound the control inputs.

The shared workspace  $P \subset \mathbb{R}^N$  is modeled as a full-dimensional polytope defined by:

$$P = \{\mathbf{x} \in \mathbb{R}^N \mid \mathbf{A}_P \mathbf{x} + \mathbf{b}_P \leq \mathbf{0}\},$$

where  $\mathbf{A}_P$  and  $\mathbf{b}_P$  specify the linear inequalities that define the polytope.

### 4.2.2 Type-A & Type-B Security

Type-A security, defined in [Definition 19](#) as an extension of single-agent path security, is verified independently for each critical agent by ensuring the existence of a valid copy path.

Type-B security, defined in [Definition 23](#), introduces coupling between agents by requiring that whenever an agent is in a secret state, another agent produces the same observation.

#### 4.2.2.1 Security Property

A global trajectory  $\mathbf{x}(t)$  is the tuple of all agents' trajectories in a bounded polytope  $P$ :  $(\mathbf{x}_1(t), \dots, \mathbf{x}_m(t))$ .

Type-A security for continuous multi-agent systems is defined as follows:

**Definition 28. Global Trajectory Type-A Security**

A global trajectory  $\mathbf{x}(t)$  is secure if there exists a global trajectory  $\mathbf{x}'(t) = (\mathbf{x}'_1(t), \dots, \mathbf{x}'_m(t))$  in  $P$  such that:

- $H(\mathbf{x}_j(t)) = H(\mathbf{x}'_j(t)), \forall t \in [0, t_f], \forall j \in A_c$ , and
- If  $\mathbf{x}_j$  is in secret states during a time interval  $[t_1, t_2]$ , where  $t_2 > t_1 \geq 0$ , i.e.,  $\mathbf{x}_j(t) \in Q_S^j, \forall t \in [t_1, t_2]$ , then  $\mathbf{x}'_j$  is in non-secret states, i.e.,  $\mathbf{x}'_j(t) \notin Q_S^j, \forall t \in [t_1, t_2], \forall j \in A_c$ .

Similarly, Type-B security for continuous multi-agent systems is defined as follows:

**Definition 29. Global Trajectory Type-B Security**

A global trajectory  $\mathbf{x}(t) = (\mathbf{x}_1(t), \dots, \mathbf{x}_m(t))$  is secure if the following holds:

- For any agent  $j \in A$  is in secret states over a time interval  $[t_1, t_2]$ , where  $t_2 > t_1 \geq 0$ , i.e.,  $\mathbf{x}_j(t) \in Q_S^j, \forall t \in [t_1, t_2]$ , then there exists at least one other agent  $l \in A$  (with  $l \neq j$ ), such that their external trajectories are identical over the same interval, i.e.,  $H(\mathbf{x}_j(t)) = H(\mathbf{x}_l(t)), \forall t \in [t_1, t_2]$ .

**4.2.2.2 Construction of Secure WTS**

To enforce either Type-A or Type-B security in continuous multi-agent systems, the same methodology as in the discrete setting is followed. Specifically:

- For Type-A security, the Global Twin-sWTS is constructed. Each agent is paired with a corresponding copy, resulting in a total of  $2m$  agents:  $m$  real agents and  $m$  copy agents.
- For Type-B security, the Global Type-B WTS is used. In this case, there are  $m$  agents, and security is achieved through observation indistinguishability between agents.

For both security types, candidate transitions in the corresponding WTS are verified by solving a discrete-time trajectory optimization problem, analogous to the one formulated in **Problem 2**. The number of agents involved differs between the two cases:

$$M = \begin{cases} 2m, & \text{for Type-A security,} \\ m, & \text{for Type-B security.} \end{cases}$$

The optimization problem is then formulated over the trajectories of all  $M$  agents as follows:

**Problem 3. Discrete-Time Global Trajectory Optimization**

$$\begin{aligned}
 \min \quad & t_f \\
 \text{s.t.} \quad & t_{cross} = n_{cross} \cdot dt_1, \quad dt_1 > 0 \\
 & t_f = t_{cross} + n_{after} \cdot dt_2, \quad dt_2 > 0 \\
 & \mathbf{x}_j(i+1) = \mathbf{x}_j(i) + dt_1 \cdot (\mathbf{A}^j \mathbf{x}_j(i) + \mathbf{B}^j \mathbf{u}_j(i) + \mathbf{b}^j), \\
 & \quad \quad \quad i = 0, \dots, n_{cross} - 1, \quad j \in M \\
 & \mathbf{x}_j(i+1) = \mathbf{x}_j(i) + dt_2 \cdot (\mathbf{A}^j \mathbf{x}_j(i) + \mathbf{B}^j \mathbf{u}_j(i) + \mathbf{b}^j), \\
 & \quad \quad \quad i = n_{cross}, \dots, n - 1, \quad j \in M \\
 & \mathbf{x}_j(0) = \mathbf{x}_{c,p_1^j}, \quad \mathbf{x}_j(n) = \mathbf{x}_{c,p_2^j}, \quad j \in M \\
 & \mathbf{u}_j(i) \in U^j, \quad i = 0, \dots, n - 1, \quad j \in M \\
 & \mathbf{A}_P \mathbf{x}_j(i) + \mathbf{b}_P \leq \mathbf{0}, \quad i = 0, \dots, n, \quad j \in M \\
 & \mathbf{H}_j \mathbf{x}_j(i) + \mathbf{g}_j < 0, \quad i = 0, \dots, n, \quad j \in M \\
 & \mathbf{a}_j^T \mathbf{x}_j(n_{cross}) + b_j = 0, \quad j \in M
 \end{aligned}$$

Note that agents are allowed to have different dynamics and individual input constraints, reflecting realistic multi-agent scenarios.

### 4.2.3 Type-C Security

Contrary to Type-A and Type-B security, Type-C secure planning requires the construction of a new WTS. This is primarily because Type-C security cannot be verified at a single point in time; instead, it must be evaluated over a time window.

Moreover, in the centroid-to-centroid motion framework adopted in this work, each transition spans a finite time interval. Specifically, during a transition, an agent remains in the current state for  $t \in [0, t_{cross})$  and in the next state for  $t \in (t_{cross}, t_f]$ . As a result, the construction of the Global Type-C Countdown WTS is more complex, compared to the case of discrete systems.

### 4.2.3.1 Security Property

The formal definition of global trajectory Type-C security is given below:

**Definition 30. Global Trajectory Type-C Security**

A global trajectory  $\mathbf{x}(t) = (\mathbf{x}_1(t), \dots, \mathbf{x}_m(t))$  is secure if there exists a constant  $k \geq 0$  such that the following holds:

- For any time  $t \geq 0$ , and any agent  $j \in A$ , if  $\mathbf{x}_j(t) \in Q_S^j$ , then there exists at least one other agent  $l \in A$  (with  $l \neq j$ ), and a time  $t' \in [t, t + k]$  such that  $\mathbf{x}_l(t') \in Q_S^l$ .

### 4.2.3.2 Construction of Secure WTS

To enforce Type-C security in infinite workspaces, a suitable Weighted Transition System (WTS) must be constructed after solving trajectory optimization problem for each candidate transition in the Global Type-C WTS

$$\mathcal{T}_g^C = (P, P_0, \rightarrow_g, w_g, \mathcal{AP}_g, L_g, A, Y_g, H_g, Q_{S,g}).$$

The feasibility of a transition between any two distinct global states is verified by checking whether a solution for **Problem 3** exists.

In the current setting, agents navigate between sub-polytope centroids, and Type-C security must be maintained. This poses a challenge: when an agent enters a secret region during its transition between centroids, the  $k$ -second countdown for satisfying the security condition begins at the moment of entry, not when the agent reaches the centroid.

A final step is required to construct a secure WTS suitable for continuous systems. This construction serves as an analog to the Global Type-C Countdown-WTS defined in **Definition 25** for discrete systems. Recall that in the discrete setting, each state of the Global Type-C Countdown-WTS captures the current position of each agent, and how much time remains in each countdown (if it is active).

However, a fundamental challenge in modeling continuous systems arises from the continuous nature of time. Since time evolves over  $\mathbb{R}_+$ , representing all possible countdown values would require an infinite number of states. To address this, a time discretization parameter  $r \in \mathbb{R}_+$  is introduced, enabling a finite representation of countdown values. Instead of allowing each of the  $m$  counters, one for every agent, to take arbitrary values in the interval  $[0, k]$ , this

interval is discretized into  $r + 1$  equally spaced values:

$$0, \frac{k}{r}, \frac{2k}{r}, \dots, \frac{(r-1)k}{r}, k$$

Each time an agent reaches the centroid of a sub-polytope with an active countdown, i.e.,  $c \in [0, k]$ , the countdown value is rounded down to the nearest smaller discretized value. This approximation is formalized by the projection function  $p : [0, k] \rightarrow \{0, \frac{k}{r}, \dots, k\}$ , defined as:

$$c' = p(c) = \left\lfloor c \cdot \frac{r}{k} \right\rfloor \cdot \frac{k}{r}.$$

This ensures safety by always underestimating the remaining countdown time, thereby making the  $k$ -second window tighter. Although this may exclude feasible secure trajectories, it guarantees security. This trade-off favors correctness over completeness. Moreover, increasing the value of  $r$  improves the fidelity of the model, allowing for a closer approximation of continuous time.

The parameter  $r$  is user-defined and should be selected to balance accuracy and computational complexity: larger values offer finer time resolution, but also increase the size of the resulting WTS, potentially leading to computational inefficiency.

A subset  $G_1 \subseteq A$  is used to identify the agents with an active countdown upon reaching the centroid of  $p_1^i$ . However, in the case of continuous systems, this set is determined by the specific values of  $\Delta t_1 = t_{cross} - 0$  and  $\Delta t_2 = t_f - t_{cross}$  for each transition.

Note that transitions of the form  $(x_1, x_2) \in \rightarrow_V$ , where  $x_1 = (p_1, c_1)$  and  $x_2 = (p_2, c_2)$ , with  $p_1 = p_2$ , are only permitted if all counters in  $c_1$  are None. In this case, all counters in  $c_2$  are also set to None.

If any counter in  $c_1$  is active, transitions with  $x_2 = x_1$  must be excluded. While these transitions neither change the robots' positions ( $x_1 = x_2$ ) nor advance any countdowns ( $c_1 = c_2$ , since no time passes), leaving them in the system can cause problems if single-state suffixes occur. In such cases, the robots remain indefinitely in the same states. Therefore, active counters eventually expire, violating Type-C security.

The Global Continuous Type-C Countdown-WTS is formally defined as:

**Definition 31. Continuous Global Type-C Countdown-WTS**

Given a Global Type-C WTS

$$\mathcal{T}_g^C = (P, P_0, \rightarrow_g, w_g, \mathcal{AP}_g, L_g, A, Q_{S,g}),$$

the Continuous Global Type-C Countdown-WTS is

$$\mathcal{T}_{g,cont}^C = (X, X_0, \rightarrow_V, w_V, \mathcal{AP}_V, L_V, A, Q_{S,g}),$$

where:

- $X \subseteq P \times \left\{0, \frac{k}{r}, \frac{2k}{r}, \dots, \frac{(r-1)k}{r}, k, None\right\}^m$  is the set of states
- $X_0 \subseteq P_0 \times \{k, None\}^m$  is the set of initial states defined as follows:

For any

$$p = (p_1, \dots, p_m) \in P_0,$$

$$\text{and } c = (c_1, c_2, \dots, c_m) \in \left\{0, \frac{k}{r}, \frac{2k}{r}, \dots, \frac{(r-1)k}{r}, k, None\right\}^m,$$

a state  $x = (p, c) \in X_0$  if the following conditions hold:

- If the number of agents  $i$  such that  $p_i \in Q_S^i$  is either zero or at least two, then for all  $i \in A$ ,  $c_i = None$ .
  - If exactly one agent  $i$  satisfies  $p_i \in Q_S^i$ , then  $c_i = k$  for that agent, and  $c_j = None$  for all  $j \neq i$ .
  - If  $k = 0$ , the number of agents in a secret state must not be exactly one; otherwise, Type-C security is violated.
- $\rightarrow_V \subseteq X \times X$  is the transition relation, defined as follows:

For any

$$p_1 = (p_1^1, p_1^2, \dots, p_1^m) \in P, \quad p_2 = (p_2^1, p_2^2, \dots, p_2^m) \in P, \quad p_1 \neq p_2,$$

$$c_1 = (c_1^1, c_1^2, \dots, c_1^m) \in \left\{0, \frac{k}{r}, \frac{2k}{r}, \dots, \frac{(r-1)k}{r}, k, None\right\}^m,$$

$$c_2 = (c_2^1, c_2^2, \dots, c_2^m) \in \left\{0, \frac{k}{r}, \frac{2k}{r}, \dots, \frac{(r-1)k}{r}, k, None\right\}^m,$$

$$x_1 = (p_1, c_1) \in X, \quad x_2 = (p_2, c_2) \in X.$$

are defined.

The transition  $(x_1, x_2) \in \rightarrow_V$  if:

–  $(p_1, p_2) \in \rightarrow_g$ , and one of the following cases holds

– **Case 1: Multiple agents in secret states at  $p_1$**

There are  $s \geq 0$  agents transitioning to secret states at  $p_2$ .

\*  $s \neq 1$ : Type-C security is maintained. The counters update:

$$c_2^i = \text{None}, \forall i \in A.$$

\*  $s = 1$ : Let  $b$  be the agent with  $p_2^b \in Q_S^b$ . It must hold that  $\Delta t_2 \leq k$ . The counters update:

$$c_2^b = p(k - \Delta t_2), \text{ and } c_2^i = \text{None}, \forall i \in A \setminus \{b\}.$$

– **Case 2: One agent  $a \in G_1$  in a secret state at  $p_1$**

There are  $s \geq 0$  agents transitioning to secret states at  $p_2$ . It must hold that  $\Delta t_1 \leq c_1^a$ .

\*  $s \geq 2$ : The counters update:  $c_2^i = \text{None}, \forall i \in A$ .

\*  $s = 1$ : Let  $b$  be the agent with  $p_2^b \in Q_S^b$ .

·  $a \neq b$ : It must hold that  $\Delta t_2 \leq k$ . The counters update:

$$c_2^b = p(k - \Delta t_2), \text{ and } c_2^i = \text{None}, \forall i \in A \setminus \{b\}.$$

·  $a = b$ : It must hold that  $t_f \leq c_1^a$ . The counters update:

$$c_2^a = p(c_1^a - t_f), \text{ and } c_2^i = \text{None}, \forall i \in A \setminus \{a\}.$$

\*  $s = 0$ : It must hold that  $t_f \leq c_1^a$ . The counters update:

$$c_2^a = p(c_1^a - t_f), \text{ and } c_2^i = \text{None}, \forall i \in A \setminus \{a\}.$$

– **Case 3: No agent in a secret state at  $p_1$ , and no agent in  $G_1$**

There are  $s \geq 0$  agents transitioning to secret states at  $p_2$ .

\*  $s \neq 1$ : Type-C security is maintained and the counters update:  $c_2^i = \text{None}, \forall i \in A$ .

\*  $s = 1$ : Let  $b$  be the agent with  $p_2^b \in Q_S^b$ . It must hold that  $\Delta t_2 \leq k$ . The counters update:

$$c_2^b = p(k - \Delta t_2), \text{ and } c_2^i = \text{None}, \forall i \in A \setminus \{b\}.$$

– **Case 4: No agent in a secret state at  $p_1$ , and one agent  $a \in G_1$**

There are  $s \geq 0$  agents transitioning to secret states at  $p_2$ .

\*  $s = 0$ : It must hold that  $t_f \leq c_1^a$ . The counters update:

$$c_2^a = p(c_1^a - t_f), \text{ and } c_2^i = \text{None}, \forall i \in A \setminus \{a\}.$$

\*  $s = 1$ : Let  $b$  be the agent with  $p_2^b \in Q_S$ .

·  $a \neq b$ : It must hold that  $\Delta t_1 \leq c_1^a$  and  $\Delta t_2 \leq k$ . The counters update:

$$c_2^b = p(k - \Delta t_2), \text{ and } c_2^i = \text{None}, \forall i \in A \setminus \{b\}.$$

·  $a = b$ : It must hold that  $t_f \leq c_1^a$ . The counters update:

$$c_2^a = p(c_1^a - \Delta t_2), \text{ and } c_2^i = \text{None}, \forall i \in A \setminus \{a\}.$$

\*  $s \geq 2$ : It must hold that  $\Delta t_1 \leq c_1^a$ . The counters update:

$$c_2^i = \text{None}, \forall i \in A.$$

$(x_1, x_2) \in \rightarrow_V$ , with  $x_1 = (p, c_1) \in X$  and  $x_2 = (p_2, c_2) \in X$ , and  $p \in P$ , only if  $c_1 = c_2 = \{\text{None}\}^m$ .

- $w_V : X \times X \rightarrow \mathbb{R}_+$  is the cost function defined as: for any  $x_1 = (p_1, c_1) \in X$ ,  $x_2 = (p_2, c_2) \in X$  such that  $(x_1, x_2) \in \rightarrow_V$ ,  $w_V(x_1, x_2) = w_g(p_1, p_2)$
- $L_V : X \rightarrow \mathcal{AP}_g$  is the labeling function defined as: for any agent  $x = (p, c) \in X$ ,  $L_V(x) = L_g(p)$
- $\mathcal{AP}_g$ ,  $A$  and  $Q_{S,g}$  are defined identically to those in  $\mathcal{T}_g^C$

**Algorithm 1** can then be applied using  $\mathcal{T}_{c,cont}^C$  as the input WTS. Once a global path on  $\mathcal{T}_{c,cont}^C$  is computed, the corresponding continuous trajectory is obtained by concatenating the optimal trajectories associated with the transitions in the path.

## 4.3 Implementation Notes

This thesis builds on the idea of partitioning the workspace using linear predicates, as introduced by Kloetzer and Belta in [16], who developed an automated control framework for single agents operating in a polytope in  $\mathbb{R}^N$  under affine dynamics, subject to LTL specifications. Their algorithm was implemented in *LTLCon* [44], a MATLAB package.

For the purposes of this thesis, a new package was developed to incorporate security guarantees for the trajectories of both single-agent and multi-agent systems while ensuring satisfaction of LTL specifications. Some utility scripts from *LTLCon* were reused for importing input data and generating plots. The implementation relies on two core tools:

- *cddmex* [45] – a MATLAB interface to the *cdd* package [46], which supports polyhedral computations, including conversions between H-representation and V-representation of polytopes. Version 1.0.1 has been used.
- *LTL2BA* [47] – a tool for efficiently translating LTL formulas into Büchi automata, based on the algorithm presented in [48]. Version 1.3 has been used.

All simulations presented in [Chapter 3](#) and [Chapter 4](#) were performed using Python 3.11, with graphical outputs generated via MATLAB R2021a, *Graphviz* and *Inkscape*. The nonlinear discrete-time optimization problem formulated in [Problem 2](#) was solved using *CasADi* version 3.7.0 [49]. Finally, small bridging scripts were written to facilitate the transition between MATLAB and Python.

All scripts were executed on a 2022 MacBook Air (M2, 16 GB RAM).

## 4.4 Motivating Examples

This Section presents examples of secure path planning for continuous single-agent and multi-agent systems, demonstrating both the correctness and practical applicability of the proposed framework. The examples focus on scenarios involving surveillance with obstacle avoidance on a manufacturing floor.

### 4.4.1 Secure Navigation of an Autonomous Robot

Consider a 2D workspace  $P$ :

$$P = \{\mathbf{x} = (x, y) \in \mathbb{R}^2 \mid 0 \leq x \leq 10 \text{ and } 0 \leq y \leq 10\},$$

which can be expressed using the notation of [Equation 4.1](#) with:

$$\mathbf{A}_P = \begin{bmatrix} -1 & 0 \\ 0 & -1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad \mathbf{b}_P = \begin{bmatrix} 0 \\ 0 \\ -10 \\ -10 \end{bmatrix}$$

This workspace models the layout of an automated manufacturing floor in a factory. A mobile robot is assigned the task of assembling electronic devices and validating their functionality. The process includes collecting tools and parts, assembling the product, and then performing functional verification at test stations, all while navigating around restricted zones and machinery.

Eight regions  $P_1, P_2, \dots, P_8$  are defined, each corresponding to specific workstations or obstacles within this environment:

$$\begin{aligned} P_1 &= \{\mathbf{x} = (x, y) \in \mathbb{R}^2 \mid 8 < x < 10 \text{ and } 8 < y < 10\} \\ P_2 &= \{\mathbf{x} = (x, y) \in \mathbb{R}^2 \mid 3 < x < 5 \text{ and } 5 < y < 8\} \\ P_3 &= \{\mathbf{x} = (x, y) \in \mathbb{R}^2 \mid 6 < x < 8 \text{ and } 5 < y < 6\} \\ P_4 &= \{\mathbf{x} = (x, y) \in \mathbb{R}^2 \mid 8 < x < 10 \text{ and } 0 < y < 2\} \\ P_5 &= \{\mathbf{x} = (x, y) \in \mathbb{R}^2 \mid 6 < x < 8 \text{ and } 2 < y < 4\} \\ P_6 &= \{\mathbf{x} = (x, y) \in \mathbb{R}^2 \mid 5 < x < 6 \text{ and } 0 < y < 2\} \\ P_7 &= \{\mathbf{x} = (x, y) \in \mathbb{R}^2 \mid 0 < x < 2 \text{ and } 2 < y < 5\} \\ P_8 &= \{\mathbf{x} = (x, y) \in \mathbb{R}^2 \mid 2 < x < 3 \text{ and } 6 < y < 8\} \end{aligned}$$

The first five regions,  $P_1$  to  $P_5$ , represent locations that the robot must visit

along its path, while the last three,  $P_6$  to  $P_8$ , correspond to obstacles that the robot must avoid. For further details, refer to [Table 4.1](#).

Region	Description
$P_1$	Assembly Station: Robot assembles the device.
$P_2$	Test Bench A: Robot tests the device, revisits frequently.
$P_3$	Test Bench B: Robot adjusts the device regularly.
$P_4$	Tool Station: Robot picks up tools.
$P_5$	Parts Storage: Robot collects parts.
$P_6$	Obstacle Zone: Robot must avoid this area.
$P_7$	Restricted Area: Access is prohibited for the robot.
$P_8$	Machinery Area: Robot must avoid machinery.

Table 4.1: Description of the regions in the workspace (Single-agent scenario)

The corresponding matrices  $\mathbf{A}_1, \dots, \mathbf{A}_8$  and vectors  $\mathbf{b}_1, \dots, \mathbf{b}_8$ , which define each region  $P_i \subset \mathbb{R}^N$  via the inequality  $\mathbf{A}_i \mathbf{x} \leq \mathbf{b}_i$  for all  $i \in \{1, \dots, 8\}$ , can be directly obtained from the linear constraints characterizing each region.

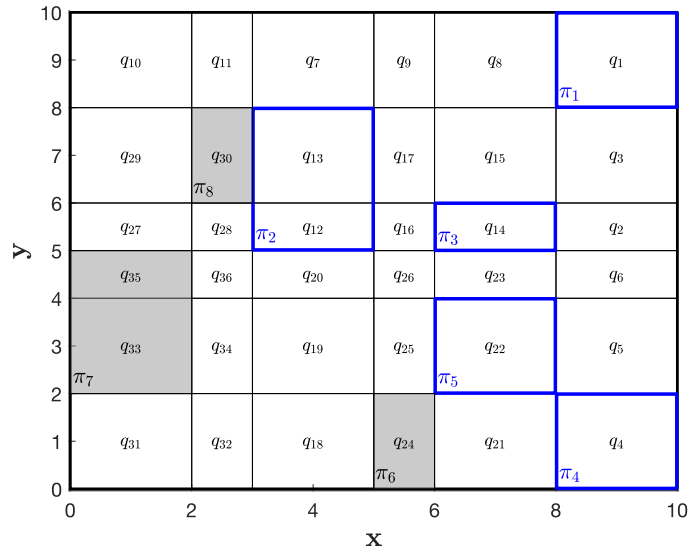


Figure 4.1: Division of the workspace  $P$  into sub-polytopes based on predicates

The combination of the predicates  $\pi_1$  through  $\pi_8$ , along with the boundaries of the polytope  $P$ , partitions the workspace into 36 disjoint sub-polytopes, as shown in [Figure 4.1](#). The sub-polytopes where predicates  $\pi_1$

through  $\pi_5$  are satisfied are outlined in blue. In contrast, regions corresponding to obstacles, where predicates  $\pi_6$  through  $\pi_8$  hold, are shaded in gray.

One initial region is introduced, one secret region, and three observation-equivalence regions, defined as follows:

$$\begin{aligned} P_{S,1} &= \{\mathbf{x} = (x, y) \in \mathbb{R}^2 \mid 5 < x < 10 \text{ and } 5 < y < 10\} \\ P_{\text{obs},1} &= \{\mathbf{x} = (x, y) \in \mathbb{R}^2 \mid 3 < x < 5 \text{ and } 0 < y < 8\} \\ P_{\text{obs},2} &= \{\mathbf{x} = (x, y) \in \mathbb{R}^2 \mid 0 < x < 10 \text{ and } 8 < y < 10\} \\ P_{\text{obs},3} &= \{\mathbf{x} = (x, y) \in \mathbb{R}^2 \mid 6 < x < 8 \text{ and } 4 < y < 6\} \\ P_{\text{init},1} &= \{\mathbf{x} = (x, y) \in \mathbb{R}^2 \mid 0 < x < 2 \text{ and } 0 < y < 2\} \end{aligned}$$

In this setup:

- $P_{S,1}$  denotes the secret region of the workspace, used to model areas where sensitive or hidden operations take place.
- $P_{\text{obs},1}$ ,  $P_{\text{obs},2}$ , and  $P_{\text{obs},3}$  are the observation-equivalence regions..
- $P_{\text{init},1}$  defines the initial region, which is the robot's origin point prior to the execution of its task.

The addition of these regions does not further subdivide the workspace beyond the 36 sub-polytopes originally defined by the combination of predicates and the workspace boundaries. The updated partitioning, including the newly introduced regions of interest, is illustrated in [Figure 4.2](#). In this figure:

- The initial region is outlined in orange,
- The secret region is outlined with a dashed red border,
- The observation-equivalence regions are lightly shaded in blue, red, and orange.

Each color-coded observation region corresponds to a distinct observation class: all states within the same shaded region produce identical observations from the intruder's perspective.

The single integrator dynamics described in [Equation 2.2](#) are used:

$$\dot{\mathbf{x}} = \mathbf{u},$$

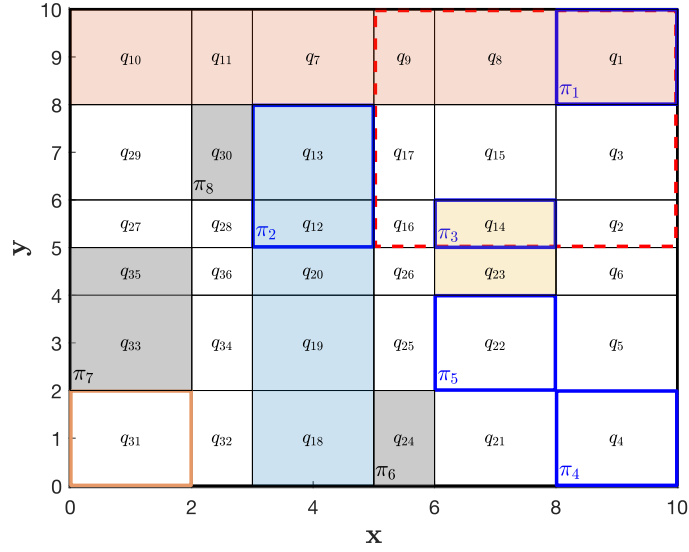


Figure 4.2: Partition of the workspace  $P$  into sub-polytopes based on predicates, initial and secret regions, and observation equivalence classes

where  $\mathbf{x} = [x \ y]^T$  and  $\mathbf{u} = [u_1 \ u_2]^T$ , subject to the input constraints:

$$-2 \leq u_1 \leq 2, \quad -2 \leq u_2 \leq 2.$$

The continuous workspace is discretized into a finite set of sub-polytopes, resulting in the set of states:

$$Q = \{q_1, q_2, \dots, q_{36}\}.$$

An initial position  $\mathbf{x}_0 = (x_0, y_0) = (1, 1)$  is selected. Since  $\mathbf{x}_0 \in P_{\text{init},1}$ , it belongs to sub-polytope  $q_{31}$ .

The transition relation includes transitions between all adjacent states (i.e., states that share a facet). In addition, each state also has a self-transition.

The set of atomic propositions is defined as:

$$\mathcal{AP} = \{\pi_1, \dots, \pi_8\},$$

and the labeling function  $L : Q \rightarrow 2^{\mathcal{AP}}$  assigns atomic propositions to specific

states:

$$\begin{aligned}
 L(q) &= \pi_1, q \in \{q_1\}, & L(q) &= \pi_2, q \in \{q_{12}, q_{13}\}, \\
 L(q) &= \pi_3, q_i \in \{q_{14}\}, & L(q) &= \pi_4, q \in \{q_4\}, \\
 L(q) &= \pi_5, q \in \{q_{22}\}, & L(q) &= \pi_6, q \in \{q_{24}\} \\
 L(q) &= \pi_7, q \in \{q_{33}, q_{35}\}, & L(q) &= \pi_8, q \in \{q_{30}\} \\
 L(q) &= \emptyset, q \in Q \setminus \{q_1, q_4, q_{12}, q_{13}, q_{14}, q_{22}, q_{24}, q_{30}, q_{33}, q_{35}\}
 \end{aligned}$$

The set of observations is defined as:

$$Y = \{o_1, o_2, o_3, o_4\},$$

and the observation function  $H : Q \rightarrow Y$  groups states into observation-equivalence classes:

$$\begin{aligned}
 H(q) &= o_1, \forall q \in Q_1 = \{q_{12}, q_{13}, q_{18}, q_{19}, q_{20}\} \\
 H(q) &= o_2, \forall q \in Q_2 = \{q_1, q_7, q_8, q_9, q_{10}, q_{11}\} \\
 H(q) &= o_3, \forall q \in Q_3 = \{q_{14}, q_{21}\} \\
 H(q) &= o_4, \forall q \in Q \setminus (Q_1 \cup Q_2 \cup Q_3)
 \end{aligned}$$

The set of secret states is defined as:

$$Q_S = \{q_1, q_2, q_3, q_8, q_9, q_{14}, q_{15}, q_{16}, q_{17}\}.$$

The robot's task is formally specified by the following LTL formula:

$$\begin{aligned}
 \phi = & \square(\diamond\pi_2 \wedge \diamond\pi_3 \wedge \neg(\pi_6 \vee \pi_7 \vee \pi_8)) \wedge \diamond\pi_1 \wedge \diamond\pi_4 \wedge \diamond\pi_5 \\
 & \wedge (\neg(\pi_1 \vee \pi_5) \mathcal{U} \pi_4) \wedge (\neg\pi_1 \mathcal{U} \pi_5)
 \end{aligned}$$

This formula encodes the following task requirements:

- $\square(\diamond\pi_2 \wedge \diamond\pi_3)$ : The robot must visit regions  $\pi_2$  (Test Bench A) and  $\pi_3$  (Test Bench B) infinitely often to ensure continuous adjustment of assembled devices.
- $\square\neg(\pi_6 \vee \pi_7 \vee \pi_8)$ : The robot must never enter restricted regions, which include heavy machinery and hazard zones.
- $\diamond\pi_1 \wedge \diamond\pi_4 \wedge \diamond\pi_5$ : Regions  $\pi_1$  (Assembly Station),  $\pi_4$  (Tool Station), and  $\pi_5$  (Parts Storage) must each be visited at least once.

- $(\neg(\pi_1 \vee \pi_5) \mathcal{U} \pi_4)$ : Region  $\pi_4$  must be visited before any visit to  $\pi_1$  or  $\pi_5$ . This ensures that the robot first collects the necessary tools ( $\pi_4$ ) before starting the assembly or gathering parts for the device.
- $(\neg\pi_1 \mathcal{U} \pi_5)$ : Region  $\pi_5$  must be visited before  $\pi_1$ . This ensures that the robot gathers all the necessary parts from  $\pi_5$  before proceeding to  $\pi_1$  for assembly.

Together, the last two “until” operators enforce the strict sequence:  $\pi_4 \rightarrow \pi_5 \rightarrow \pi_1$  ensuring the robot collects tools before gathering parts, and both before assembly.

Using the secure path planning procedure from [Algorithm 2](#), the following discrete real path is generated. It has a duration of 24.75 seconds and is given by:

$$\begin{aligned}
 & q_{31} \rightarrow q_{32} \rightarrow q_{34} \rightarrow q_{36} \rightarrow q_{28} \rightarrow q_{12} \rightarrow q_{13} \rightarrow q_{17} \rightarrow q_{15} \\
 & \rightarrow q_3 \rightarrow q_2 \rightarrow q_6 \rightarrow q_5 \rightarrow q_4 \rightarrow q_5 \rightarrow q_{22} \rightarrow q_5 \rightarrow q_6 \rightarrow q_2 \\
 & \rightarrow q_3 \rightarrow q_1 \rightarrow q_3 \rightarrow q_{15} \rightarrow q_{17} \rightarrow q_{13} \rightarrow q_{12} \rightarrow q_{16} \\
 & \rightarrow (q_{14} \rightarrow q_{16} \rightarrow q_{12} \rightarrow q_{16})^\omega
 \end{aligned}$$

The corresponding copy path is:

$$\begin{aligned}
 & q_{31} \rightarrow q_{31} \rightarrow q_{32} \rightarrow q_{32} \rightarrow q_{32} \rightarrow q_{18} \rightarrow q_{18} \rightarrow q_{32} \rightarrow q_{32} \\
 & \rightarrow q_{34} \rightarrow q_{34} \rightarrow q_{34} \rightarrow q_{34} \rightarrow q_{36} \rightarrow q_{28} \rightarrow q_{27} \rightarrow q_{29} \rightarrow q_{29} \rightarrow q_{29} \\
 & \rightarrow q_{29} \rightarrow q_{10} \rightarrow q_{29} \rightarrow q_{27} \rightarrow q_{28} \rightarrow q_{12} \rightarrow q_{20} \rightarrow q_{26} \\
 & \rightarrow (q_{23} \rightarrow q_{26} \rightarrow q_{20} \rightarrow q_{26})^\omega
 \end{aligned}$$

A video demonstrating the evolution of the real and copy trajectories is available at: <https://drive.google.com/file/d/1HGDdAJy64qz2wBPb9THLib-tNJmsZjoi/view?usp=sharing>. The real trajectory is depicted in green, while the copy trajectory is shown in light green with a black outline.

#### 4.4.2 Secure Navigation of a Multi-Agent Robot Team

Consider a continuous system of two agents operating in a bounded 2D workspace:

$$P' = \{\mathbf{x} = (x, y) \in \mathbb{R}^2 \mid 0 \leq x \leq 6 \text{ and } 0 \leq y \leq 6\},$$

The workspace is partitioned into several semantically meaningful regions associated with atomic propositions. The task-related regions are defined as:

$$P_1 = \{\mathbf{x} = (x, y) \in \mathbb{R}^2 \mid 0 < x < 3 \text{ and } 4 < y < 6\}$$

$$P_2 = \{\mathbf{x} = (x, y) \in \mathbb{R}^2 \mid 0 < x < 3 \text{ and } 0 < y < 2\}$$

$$P_3 = \{\mathbf{x} = (x, y) \in \mathbb{R}^2 \mid 0 < x < 3 \text{ and } 2 < y < 4\}$$

These regions are summarized in [Table 4.2](#).

Region	Description
$P_1$	Assembly Station: Robots assemble the device.
$P_2$	Parts Storage: Robots collect parts.
$P_3$	Obstacle Zone: Robots must avoid this area.

Table 4.2: Description of the regions in the workspace (Multi-agent scenario)

Additionally, one initial region is defined, one secret region, and two observation-equivalence regions:

$$P_{S,1} = \{\mathbf{x} = (x, y) \in \mathbb{R}^2 \mid 3 < x < 6 \text{ and } 2 < y < 4\}$$

$$P_{\text{obs},1} = \{\mathbf{x} = (x, y) \in \mathbb{R}^2 \mid 0 < x < 3 \text{ and } 0 < y < 6\}$$

$$P_{\text{obs},2} = \{\mathbf{x} = (x, y) \in \mathbb{R}^2 \mid 3 < x < 6 \text{ and } 0 < y < 6\}$$

$$P_{\text{init},1} = \{\mathbf{x} = (x, y) \in \mathbb{R}^2 \mid 3 < x < 6 \text{ and } 4 < y < 6\}$$

The partitioned workspace is illustrated in [Figure 4.3](#). States in the left column ( $0 < x < 3$ ) share the same observation, and, similarly, states in the right column ( $3 < x < 6$ ) form a second observation-equivalence class. These equivalence classes are not explicitly highlighted in the figure to maintain visual clarity.

The task for the two robots is specified by the following LTL formula:

$$\begin{aligned} \phi_1 = & \square \diamond (\pi_1^1 \wedge \pi_1^2) \wedge \diamond (\pi_2^1 \wedge \neg \pi_2^2) \wedge \diamond (\pi_2^2 \wedge \neg \pi_2^1) \\ & \wedge (\neg \pi_2^2 \mathcal{U} \square \pi_1^1) \wedge \square \neg (\pi_3^1 \vee \pi_3^2) \end{aligned}$$

This specification requires:

- Both robots must visit the Assembly Station infinitely often together.
- Each robot must visit the Parts Storage without the other being present. This prevents interference between the robots in a limited space.

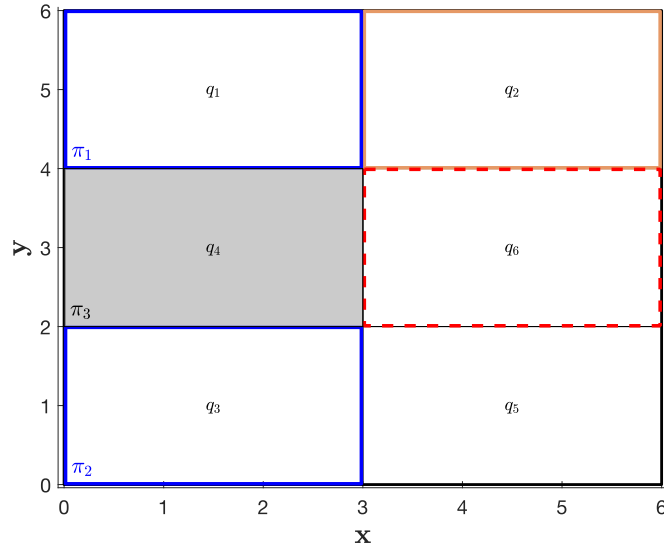


Figure 4.3: Partition of the workspace  $P'$  into sub-polytopes based on predicates, initial and secret regions, and observation equivalence classes

- Robot 2 is only allowed to retrieve parts from Storage after Robot 1 has reached and remains at the Assembly Station. This is because Robot 1 must prepare the assembly setup before Robot 2 delivers parts.
- Both robots must avoid the obstacle region at all times.

Both Robot 1 and Robot 2 operate in the workspace  $P'$  under single integrator dynamics, with control inputs bounded by  $-2 \leq u_1, u_2 \leq 2$ . Their initial positions are  $\mathbf{x}_0^1 = \mathbf{x}_0^2 = (4.5, 5)$ .

In this example, both agents are critical for Type-A security. The resulting Type-A secure path has a duration of 13.5 seconds and is given by:

$$(q_2, q_2) \rightarrow (q_6, q_2) \rightarrow (q_5, q_2) \rightarrow (q_3, q_6) \rightarrow (q_5, q_5) \rightarrow (q_6, q_5) \\ \rightarrow (q_2, q_5) \rightarrow (q_1, q_3) \rightarrow (q_1, q_5) \rightarrow (q_1, q_6) \rightarrow (q_1, q_2) \rightarrow [(q_1, q_1)]^\omega$$

The corresponding copy path is:

$$(q_2, q_2) \rightarrow (q_2, q_2) \rightarrow (q_2, q_2) \rightarrow (q_1, q_2) \rightarrow (q_2, q_2) \rightarrow (q_2, q_2) \\ \rightarrow (q_2, q_2) \rightarrow (q_1, q_1) \rightarrow (q_1, q_2) \rightarrow (q_1, q_2) \rightarrow (q_1, q_2) \rightarrow [(q_1, q_1)]^\omega$$

A video illustrating the evolution of the real and copy trajectories for Robot 1 and Robot 2 is available at: <https://drive.google.com/file/d/>

[118-OHhUVKjNXvQdLaLsyX01fq9BNr3yy/view?usp=sharing](https://www.youtube.com/watch?v=118-OHhUVKjNXvQdLaLsyX01fq9BNr3yy/view?usp=sharing). In the video, the real trajectories of Robot 1 and Robot 2 are depicted in green and blue, respectively, while the copy trajectories are shown in light green and light blue with a black outline.

Interestingly, no path satisfies Type-B security. This is due to the constraint that Robot 2 can only access the Parts Storage after Robot 1 is permanently in the Assembly Station. As a result, when Robot 2 passes from the secret region to reach the Assembly Station, Robot 1 cannot return to the right column to produce an identical observation. Consequently, the robots cannot complete the task while maintaining Type-B security.

Similarly, no path satisfies Type-C security for any value of  $k$ . After Robot 1 settles in the Assembly Station, Robot 2 must visit the Parts Storage and then return to the Assembly Station. However, doing so requires Robot 2 to pass through the secret region, which triggers the countdown. Since Robot 1 cannot leave the Assembly Station once it has entered, nor visit the secret region, Robot 2's countdown expires for any value of  $k$ , compromising Type-C security.

If the task is relaxed to the following LTL formula:

$$\phi_2 = \Box\Diamond(\pi_1^1 \wedge \pi_1^2) \wedge \Diamond(\pi_2^1 \wedge \neg\pi_2^2) \wedge \Diamond(\pi_2^2 \wedge \neg\pi_2^1) \wedge \Box\neg(\pi_3^1 \vee \pi_3^2),$$

that is, removing  $(\neg\pi_2^2 \mathcal{U} \Box\pi_1^1)$ , then Type-A, Type-B and Type-C security yield paths with identical cost.

For Type-A security, the real path is:

$$\begin{aligned} & (q_2, q_2) \rightarrow (q_2, q_6) \rightarrow (q_6, q_5) \rightarrow (q_5, q_3) \rightarrow (q_3, q_5) \rightarrow (q_5, q_6) \\ & \rightarrow (q_6, q_2) \rightarrow (q_2, q_2) \rightarrow [(q_1, q_1)]^\omega, \end{aligned}$$

while the the corresponding copy path is:

$$\begin{aligned} & (q_2, q_2) \rightarrow (q_2, q_2) \rightarrow (q_2, q_2) \rightarrow (q_2, q_1) \rightarrow (q_1, q_2) \rightarrow (q_2, q_2) \\ & \rightarrow (q_2, q_2) \rightarrow (q_2, q_2) \rightarrow [(q_1, q_1)]^\omega \end{aligned}$$

For Type-B security, the path is:

$$\begin{aligned} & (q_2, q_2) \rightarrow (q_6, q_6) \rightarrow (q_5, q_6) \rightarrow (q_3, q_5) \rightarrow (q_5, q_3) \rightarrow (q_6, q_5) \\ & \rightarrow (q_6, q_6) \rightarrow (q_2, q_2) \rightarrow [(q_1, q_1)]^\omega \end{aligned}$$

For Type-C security, with any  $k \geq 0$  and any discretization parameter

$r \geq 1$ , a secure path is found:

$$\begin{aligned} & (q_2, q_2) \rightarrow (q_6, q_6) \rightarrow (q_5, q_5) \rightarrow (q_3, q_5) \rightarrow (q_5, q_3) \rightarrow (q_5, q_5) \\ & \rightarrow (q_6, q_6) \rightarrow (q_2, q_2) \rightarrow [(q_1, q_1)]^\omega \end{aligned}$$

Note that the solver returns different paths for Type-A, Type-B, and Type-C security due to negligible numerical differences in cost, even though all paths have the same duration: 10 seconds.

Trajectories for each security type are illustrated in the following videos:

- Type-A security: <https://drive.google.com/file/d/1x8OYA1jpkz1szoyEPm69YTWV3Nenrroh/view?usp=sharing>.

Real trajectories of Robot 1 and Robot 2 are shown in green and blue, respectively; copy trajectories appear in light green and light blue with a black outline.

- For Type-B security, the trajectories are available at: [https://drive.google.com/file/d/1qeTu\\_XmNs\\_-1bDWh-KZ-cVNr8\\_RYk8c8/view?usp=sharing](https://drive.google.com/file/d/1qeTu_XmNs_-1bDWh-KZ-cVNr8_RYk8c8/view?usp=sharing).

Robot 1 and Robot 2 trajectories are depicted in green and blue, respectively.

- For Type-C security, the trajectories are available at: [https://drive.google.com/file/d/1NbCC46JNNnV\\_kD31nIUUuLZRR9dQWa\\_e/view?usp=sharing](https://drive.google.com/file/d/1NbCC46JNNnV_kD31nIUUuLZRR9dQWa_e/view?usp=sharing).

As with Type-B, Robot 1 and Robot 2 trajectories are shown in green and blue, respectively.



## **Chapter 5**

# **Conclusions and Future Work**

## 5.1 Conclusions

This thesis presents a secure-by-construction framework for planning and control in multi-agent systems with Linear Temporal Logic (LTL) specifications. The proposed methodologies address a critical gap in the literature by integrating security considerations directly into the controller synthesis process, ensuring both LTL task satisfaction and security against inference attacks from passive intruders.

The thesis builds upon abstraction-based techniques, including the construction of Weighted Transition Systems (WTS), Global WTS, and various forms of secure WTS, tailored to specific security properties and the intruder's capabilities. The framework offers formal guarantees for the resulting secure WTS. Specifically, in the discrete systems case, it has been proven that no feasible secure paths are excluded (completeness) and all retained paths are secure (correctness). For continuous systems, correctness is preserved, but completeness is compromised due to three necessary assumptions: *(i)* agents move only between region centroids, *(ii)* continuous-time dynamics are approximated using a discrete-time model, and *(iii)* all agents must synchronize when transitioning between regions. These assumptions are essential to guarantee LTL task satisfaction while preserving the desired security requirements, which is the primary objective of the thesis.

Regarding LTL task satisfaction, this work follows standard techniques from the literature, utilizing Büchi Automata and Product Büchi Automata (PBA), and computing prefix-suffix paths over the PBA.

The proposed framework is validated on both single-agent and multi-agent systems. For multi-agent systems, three distinct security notions are introduced: *(i)* Type-A (path indistinguishability from the intruder's perspective), *(ii)* Type-B (decoy-based security strategy), and *(iii)* Type-C (security accounting for possibly delayed intruder observations). Their applicability is validated through simulations in industrial-inspired scenarios, such as factory robot navigation.

The main contribution of the thesis lies in extending secure-by-construction planning with LTL task satisfaction from discrete systems to continuous systems with affine dynamics. This is achieved through workspace partitioning and centroid-based robot navigation, under constraints introduced to ensure security and task satisfaction. The results demonstrate that theoretical guarantees can be effectively translated into frameworks applicable to real-world systems, bridging the gap between formal methods and practical deployment.

## 5.2 Future Work

Several directions for future work arise from this thesis:

- **Extension to Nonlinear Dynamics:** While the current formulation assumes first-order dynamics and, specifically, all the examples use single integrator dynamics, for simplicity and tractability, many robotic systems exhibit nonlinear dynamics. Extending the control synthesis framework to such nonlinear dynamics would broaden the applicability of the approach to more practical scenarios.
- **Beyond Centroid-to-Centroid Transitions:** The current abstraction assumes that agents transition between centroids of partitioned polytopes. However, this may impose unnecessary constraints in applications where agents can move freely in the workspace. Future work could investigate more flexible abstractions to capture more trajectories while preserving formal guarantees.
- **Decentralized and Scalable Implementations:** The computational complexity of centralized planning grows quickly with the number of agents and abstraction granularity. A promising extension is to investigate decentralized or hierarchical architectures that retain correctness and security while reducing computational complexity.
- **Security against Stronger Intruders:** The thesis assumes a passive intruder with limited observability. Future work could consider more capable intruders, such as those with memory, or partial control over certain agents, and extend security formulations accordingly.
- **Experimental Validation and Human-Robot Interaction:** While the presented methods are validated in simulated industrial settings, applying them to real-world systems, especially those involving human-robot collaboration, would help validate the assumptions and uncover practical constraints not captured in the formal models.
- **Probabilistic Extensions:** Finally, incorporating stochastic models for both system dynamics and intruder observations can increase robustness. Formulations based on probabilistic opacity would enable safer operation in uncertain environments.



# References

- [1] S. Liu, A. Trivedi, X. Yin, and M. Zamani, “Secure-by-Construction Synthesis of Cyber-Physical Systems,” *Annual Reviews in Control*, vol. 53, pp. 30–50, 2022. doi: [10.1016/j.arcontrol.2022.03.004](https://doi.org/10.1016/j.arcontrol.2022.03.004).
- [2] S. M. LaValle, *Planning Algorithms*. Cambridge University Press, 2006.
- [3] E. W. Dijkstra, “A Note on Two Problems in Connexion with Graphs,” *Numerische Mathematik*, vol. 1, pp. 269–271, 1959. doi: [10.1007/BF01386390](https://doi.org/10.1007/BF01386390).
- [4] P. E. Hart, N. J. Nilsson, and B. Raphael, “A Formal Basis for the Heuristic Determination of Minimum Cost Paths,” *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968. doi: [10.1109/TSSC.1968.300136](https://doi.org/10.1109/TSSC.1968.300136).
- [5] S. Karaman and E. Frazzoli, “Sampling-Based Algorithms for Optimal Motion Planning,” *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011. doi: [10.1177/0278364911406761](https://doi.org/10.1177/0278364911406761).
- [6] C. Belta, B. Yordanov, and E. Aydin Gol, *Formal Methods for Discrete-Time Dynamical Systems*. Springer International Publishing AG, 2017.
- [7] A. Pnueli, “The Temporal Logic of Programs,” in *18<sup>th</sup> Annual Symposium on Foundations of Computer Science*, 1977, pp. 46–57. doi: [10.1109/SFCS.1977.32](https://doi.org/10.1109/SFCS.1977.32).
- [8] G. De Giacomo and M. Y. Vardi, “Automata-Theoretic Approach to Planning for Temporally Extended Goals,” in *5<sup>th</sup> European Conference on Planning*, vol. 1809, 1999, pp. 226–238. doi: [10.1007/10720246\\_18](https://doi.org/10.1007/10720246_18).

- [9] S. L. Smith, J. Tumova, C. Belta, and D. Rus, “Optimal Path Planning for Surveillance with Temporal-Logic Constraints,” *The International Journal of Robotics Research*, vol. 30, no. 14, pp. 1695–1708, 2011. DOI: [10.1177/0278364911417911](https://doi.org/10.1177/0278364911417911).
- [10] M. Kloetzer, X. C. Ding, and C. Belta, “Multi-Robot Deployment from LTL Specifications with Reduced Communication,” in *50<sup>th</sup> IEEE Conference on Decision and Control and European Control Conference*, 2011, pp. 4867–4872. DOI: [10.1109/CDC.2011.6160478](https://doi.org/10.1109/CDC.2011.6160478).
- [11] A. Ulusoy, S. S. L., X. C. Ding, C. Belta, and R. D., “Optimality and Robustness in Multi-Robot Path Planning with Temporal Logic Constraints,” *The International Journal of Robotics Research*, vol. 32, no. 8, pp. 889–911, 2013. DOI: [10.1177/0278364913487931](https://doi.org/10.1177/0278364913487931).
- [12] J. Tumova and D. V. Dimarogonas, “A Receding Horizon Approach to Multi-Agent Planning from Local LTL Specifications,” in *2014 American Control Conference*, 2014, pp. 1775–1780. DOI: [10.1109/ACC.2014.6859413](https://doi.org/10.1109/ACC.2014.6859413).
- [13] Y. Zheng, Z. Liu, A. Lai, X. Yu, and W. Lan, “Optimal Multi-agent Collision-Free Path Planning with Temporal Logic Constraints,” in *Proceedings of 2021 5<sup>th</sup> Chinese Conference on Swarm Intelligence and Cooperative Control. Lecture Notes in Electrical Engineering*, vol. 934, 2023, pp. 1468–1477. DOI: [10.1007/978-981-19-3998-3\\_137](https://doi.org/10.1007/978-981-19-3998-3_137).
- [14] C. Belta, V. Isler, and G. Pappas, “Discrete abstractions for robot motion planning and control in polygonal environments,” *IEEE Transactions on Robotics*, vol. 21, no. 5, pp. 864–874, 2005. DOI: [10.1109/TRO.2005.851359](https://doi.org/10.1109/TRO.2005.851359).
- [15] F. G. E., A. Girard, H. Kress-Gazit, and G. J. Pappas, “Temporal Logic Motion Planning for Dynamic Robots,” *Automatica*, vol. 45, pp. 343–352, 2009. DOI: [10.1016/j.automatica.2008.08.008](https://doi.org/10.1016/j.automatica.2008.08.008).
- [16] M. Kloetzer and C. Belta, “A fully automated framework for control of linear systems from temporal logic specifications,” *IEEE Transactions on Automatic Control*, vol. 53, no. 1, pp. 287–297, 2008. DOI: [10.1109/TAC.2007.914952](https://doi.org/10.1109/TAC.2007.914952).
- [17] S. Karaman, R. G. Sanfelice, and E. Frazzoli, “Optimal Control of Mixed Logical Dynamical Systems with Linear Temporal Logic Specifications,” in *47<sup>th</sup> IEEE Conference on Decision and Control*, 2008, pp. 2117–2122. DOI: [10.1109/CDC.2008.4739370](https://doi.org/10.1109/CDC.2008.4739370).

- [18] H. Kress-Gazit, G. Fainekos, and G. J. Pappas, “Temporal-Logic-Based Reactive Mission and Motion Planning,” *IEEE Transactions on Robotics*, vol. 25, no. 6, pp. 1370–1381, 2009. doi: [10.1109/TRO.2009.2030225](https://doi.org/10.1109/TRO.2009.2030225).
- [19] M. Guo and D. Dimarogonas, “Multi-Agent Plan Reconfiguration under Local LTL Specifications,” *International Journal of Robotics Research*, vol. 34, no. 2, pp. 218–235, 2014. doi: [10.1177/0278364914546174](https://doi.org/10.1177/0278364914546174).
- [20] M. Kloetzer and C. Belta, “Temporal Logic Planning and Control of Robotic Swarms by Hierarchical Abstractions,” *IEEE Transactions on Robotics*, vol. 23, no. 2, pp. 320–330, 2007. doi: [10.1109/TRO.2006.889492](https://doi.org/10.1109/TRO.2006.889492).
- [21] S. G. Loizou and K. J. Kyriakopoulos, “Automatic Synthesis of Multi-Agent Motion Tasks Based on LTL Specifications,” in *43<sup>rd</sup> IEEE Conference on Decision and Control*, vol. 1, 2004, pp. 153–158. doi: [10.1109/CDC.2004.1428622](https://doi.org/10.1109/CDC.2004.1428622).
- [22] J. W. Bryans, M. Koutny, L. Mazaré, and P. Ryan, “Opacity Generalised to Transition Systems,” in *Formal Aspects in Security and Trust. Lecture Notes in Computer Science*, vol. 3866, 2006, pp. 81–95. doi: [10.1007/11679219\\_7](https://doi.org/10.1007/11679219_7).
- [23] F. Lin, “Opacity of Discrete Event Systems and Its Applications,” *Automatica*, vol. 47, pp. 496–503, 2011. doi: [10.1016/j.automatica.2011.01.002](https://doi.org/10.1016/j.automatica.2011.01.002).
- [24] A. Saboori and C. N. Hadjicostis, “Notions of Security and Opacity in Discrete Event Systems,” in *46<sup>th</sup> IEEE Conference on Decision and Control*, 2007, pp. 5056–5061. doi: [10.1109/CDC.2007.4434515](https://doi.org/10.1109/CDC.2007.4434515).
- [25] S. Yang and X. Yin, “Secure Your Intention: On Notions of Pre-Opacity in Discrete-Event Systems,” *IEEE Transactions on Automatic Control*, vol. 68, no. 8, pp. 4754–4766, 2023. doi: [10.1109/TAC.2022.3210148](https://doi.org/10.1109/TAC.2022.3210148).
- [26] S. Lafortune, F. Lin, and C. N. Hadjicostis, “On the History of Diagnosability and Opacity in Discrete Event Systems,” *Annual Reviews in Control*, vol. 45, pp. 257–266, 2018. doi: [10.1016/j.arcontrol.2018.04.002](https://doi.org/10.1016/j.arcontrol.2018.04.002).

- [27] Y. Wu and S. Lafortune, “Comparative Analysis of Related Notions of Opacity in Centralized and Coordinated Architectures,” *Discrete Event Dynamic Systems*, vol. 23, pp. 307–339, 2013. doi: [10.1007/s10626-012-0145-z](https://doi.org/10.1007/s10626-012-0145-z).
- [28] X. Yin, M. Zamani, and S. Liu, “On Approximate Opacity of Cyber-Physical Systems,” *IEEE Transactions on Automatic Control*, vol. 66, no. 4, pp. 1630–1645, 2021. doi: [10.1109/TAC.2020.2998733](https://doi.org/10.1109/TAC.2020.2998733).
- [29] B. Zhong, S. Liu, M. Caccamo, and M. Zamani, “Secure-by-Construction Synthesis for Control Systems,” *IEEE Transactions on Automatic Control*, vol. 70, no. 6, pp. 4170–4177, 2025. doi: [10.1109/TAC.2025.3532541](https://doi.org/10.1109/TAC.2025.3532541).
- [30] S. Tasdighi Kalat, S. Liu, and M. Zamani, “Modular Verification of Opacity for Interconnected Control Systems via Barrier Certificates,” *IEEE Control Systems Letters*, vol. 6, pp. 890–895, 2022. doi: [10.1109/LCSYS.2021.3087103](https://doi.org/10.1109/LCSYS.2021.3087103).
- [31] L. Siyuan, A. Swikir, and M. Zamani, “Verification of Approximate Opacity for Switched Systems: A Compositional Approach,” *Nonlinear Analysis: Hybrid Systems*, vol. 42, p. 101 084, 2021. doi: [10.1016/j.nahs.2021.101084](https://doi.org/10.1016/j.nahs.2021.101084).
- [32] A. Saboori and C. N. Hadjicostis, “Verification of  $K$ -Step and Analysis of Its Complexity,” *IEEE Transactions on Automation Science and Engineering*, vol. 8, no. 3, pp. 549–559, 2011. doi: [10.1109/TASE.2011.2106775](https://doi.org/10.1109/TASE.2011.2106775).
- [33] A. Saboori and C. N. Hadjicostis, “Verification of Infinite-Step Opacity and Complexity Considerations,” *IEEE Transactions on Automatic Control*, vol. 57, no. 5, pp. 1265–1269, 2012. doi: [10.1109/TAC.2011.2173774](https://doi.org/10.1109/TAC.2011.2173774).
- [34] K. Zhang, X. Yin, and M. Zamani, “Opacity of Nondeterministic Transition Systems: A (Bi)Simulation Relation Approach,” *IEEE Transactions on Automatic Control*, vol. 64, no. 12, pp. 5116–5123, 2019. doi: [10.1109/TAC.2019.2908726](https://doi.org/10.1109/TAC.2019.2908726).
- [35] C. N. Hadjicostis, “Trajectory Planning under Current-State Opacity Constraints,” in *IFAC-PapersOnLine*, vol. 51, 2018, pp. 337–342. doi: [10.1016/j.ifacol.2018.06.322](https://doi.org/10.1016/j.ifacol.2018.06.322).

- [36] S. Yang, X. Yin, S. Li, and M. Zamani, “Secure-by-Construction Optimal Path Planning for Linear Temporal Logic Tasks,” in *59<sup>th</sup> IEEE Conference on Decision and Control (CDC)*, 2020, pp. 4460–4466. DOI: [10.1109/CDC42340.2020.9304153](https://doi.org/10.1109/CDC42340.2020.9304153).
- [37] X. Yu, X. Yin, S. Li, and Z. Li, “Security-Preserving Multi-Agent Coordination for Complex Temporal Logic Tasks,” *Control Engineering Practice*, vol. 123, Article 105130, 2022. DOI: [10.1016/j.conengprac.2022.105130](https://doi.org/10.1016/j.conengprac.2022.105130).
- [38] B. Cui, K. Zhu, S. Li, and X. Yin, “Security-Aware Reinforcement Learning under Linear Temporal Logic Specifications,” in *2023 IEEE International Conference on Robotics and Automation*, 2023, pp. 12 367–12 373. DOI: [10.1109/ICRA48891.2023.10160753](https://doi.org/10.1109/ICRA48891.2023.10160753).
- [39] C. Baier and J. Katoen, *Principles of Model Checking*. MIT Press, 2008.
- [40] P. Wolper, M. Y. Vardi, and A. P. Sistla, “Reasoning About Infinite Computation Paths,” in *24<sup>th</sup> Annual Symposium on Foundations of Computer Science*, 1983, pp. 185–194. DOI: [10.1109/SFCS.1983.51](https://doi.org/10.1109/SFCS.1983.51).
- [41] J. Büchi, “Symposium on Decision Problems: On a Decision Method in Restricted Second Order Arithmetic,” in *Logic, Methodology and Philosophy of Science*, ser. *Studies in Logic and the Foundations of Mathematics*, vol. 44, Elsevier, 1966, pp. 1–11. DOI: [10.1016/S0049-237X\(09\)70564-6](https://doi.org/10.1016/S0049-237X(09)70564-6).
- [42] G. Holzmann, *The SPIN Model Checker: Primer and Reference Manual*. Addison-Wesley, 2004.
- [43] A. Saboori and C. N. Hadjicostis, “Verification of Initial-State Opacity in Security Applications of Discrete Event Systems,” *Information Sciences*, vol. 246, pp. 115–132, 2013. DOI: [10.1016/j.ins.2013.05.033](https://doi.org/10.1016/j.ins.2013.05.033).
- [44] M. Kloetzer and C. Belta, LTLCon, <https://calinbelta.com/software/>, Accessed: 3 April 2025, 2005.
- [45] F. Torrisi, M. Baotic, M. Kvasnica, and M. Herceg, Mex interface to CDD (Version 1.0.1), <https://www.tbxmanager.com/version/view/106>, Accessed: 3 April 2025, 2013.
- [46] K. Fukuda, cddlib-0.94g, [https://people.inf.ethz.ch/fukudak/cdd\\_home/](https://people.inf.ethz.ch/fukudak/cdd_home/), Accessed: 3 April 2025, 2012.

- [47] P. Gastin and D. Oddoux, LTL2BA (Version 1.3), <http://www.lsv.fr/~gastin/ltl2ba/>, Accessed: 3 April 2025, 2020.
- [48] P. Gastin and D. Oddoux, “Fast LTL to Büchi Automata Translation,” in *13<sup>th</sup> Computer Aided Verification*, 2001, pp. 53–65. doi: [10.1007/3-540-44585-4\\_6](https://doi.org/10.1007/3-540-44585-4_6).
- [49] J. A. E. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, “CasADi – A Software Framework for Nonlinear Optimization and Optimal Control,” *Mathematical Programming Computation*, vol. 11, no. 1, pp. 1–36, 2019. doi: [10.1007/s12532-018-0139-4](https://doi.org/10.1007/s12532-018-0139-4).



TRITA-EECS-EX-2025:952  
Stockholm, Sweden 2025

[www.kth.se](http://www.kth.se)