Cyprus University of Technology
Department of Mechanical Engineering and Materials Science and Engineering

Doctoral Dissertation

Task Planning and Control Synthesis for Multi-Agent Systems

Anatoli Tziola

Limassol, October 2024

CYPRUS UNIVERSITY OF TECHNOLOGY

Faculty of Engineering and Technology

Department of Mechanical Engineering and Materials Science and Engineering

Doctoral Dissertation

Task Planning and Control Synthesis for Multi-Agent Systems

Anatoli Tziola

Advisor:

Savvas G. Loizou

Associate Professor

Limassol, October 2024

Advisory Committee

Doctoral Dissertation

Task Planning and Control Synthesis for Multi-Agent Systems

Presented by

Anatoli Tziola

Advisor: Savvas G. Loizou, Associate Professor
Member of the committee: Christoforos N. Hadjicostis, Professor
Member of the committee: Anastasis V. Georgiades, Associate Professor

Cyprus University of Technology
Limassol, October 2024

Copyrights

The approval of the dissertation by the Department of Mechanical Engineering and Materials Science and Engineering does not necessarily imply the approval by the Department of the views of the writer.

Acknowledgments

I would like to express my gratitude to my advisor Assoc. Prof. Savvas G. Loizou for his mentorship, guidance and support throughout the years of my PhD. He motivated and inspired me while he gave me the freedom to steer the main topic of my PhD. It was a great opportunity to conduct my doctoral studies under his supervision and learn from his expertise in multiple domains.

I am incredibly grateful for my PhD thesis committee members: Prof. Christoforos N. Hadjicostis and Assoc. Prof. Anastasis V. Georgiades for their insightful comments and recommendations that have significantly enhanced the quality of my dissertation.

I would like to thank Dr. George Georgiades for his support during the first years of my studies and for the collaboration in the projects that we were working together.

I would like to thank the European Commission for partially supporting my research through the European Union's research and innovation programs under the grant agreements 767642 (Logistics for Manufacturing (L4MS) / H2020), 951813 (Better Factory / H2020) and 101092295 (CIRCULOOS / Horizon Europe).

I would like to gratefully thank above all my Mother for Her endless love and support. I dedicate this dissertation to Her, since this dissertation would not have been possible without Her.

I would like to greatly thank all my family for giving me the complete freedom to steer my career and supporting me in every step of my life as well as during all stages of my PhD journey.

Anatoli Tziola

Limassol, October 2024

List of Publications

Journal Articles

J.1 A. A. Tziola, S. G. Loizou, "A Formal Framework for Multi-Agent Task Planning", (under review (2nd stage), IEEE Transactions on Automatic Control).

J.2 A. A. Tziola, S. G. Loizou, "Supervisory Control Synthesis for Multi-Agent Systems with Failure-Mode Reconfiguration", (under preparation).

Conference Papers

C.1 A. A. Tziola, S. G. Loizou, "Autonomous Task Planning for Heterogeneous Multi-Agent Systems", 2023 IEEE International Conference of Robotics and Automation (ICRA23), London, UK, May, 2023.

C.2 A. A. Tziola, S. G. Loizou, "Manufacturing Logistics Optimization using the SPECTER Task Planner: A Shoe Manufacturing Logistics Case Study", European Robotics Forum 2024 (ERF2024), Rimini, Italy, March, 2024.

C.3 A. A. Tziola, S. G. Loizou, "Discrete Abstractions for Manufacturing Logistics Optimization for the Food Service Industry", 2024 IEEE International Conference on Control, Decision and Information Technologies (CoDiT 2024), Valletta, Malta, July, 2024.

C.4 A. A. Tziola, S. G. Loizou, "Multi-Agent Control Synthesis with Reactive Failure-Mode based Reconfiguration", (under review, European Control Conference (ECC)).

Pre-Prints

PP.1 A. A. Tziola and S. G. Loizou, "Autonomous Task Planning for Heterogeneous Multi-Agent Systems," arXiv Preprint arXiv:2209.08611, 2022.

ABSTRACT

One of the major areas of research interest in the field of robotics has been the operation of autonomous multi-agent systems in order to perform complicated tasks in dynamical environments. Additionally, the combination of task planning with supervisory control for autonomous multi-agent systems is very challenging and it has been a topic of increasing interest in the last decade, since it can provide a robust framework for autonomous multi-agent task planning and execution. While task planning considers the determination of discrete sequences of actions that are appropriately executed ensuring that the system reaches given objectives, the supervisory control considers the orchestration of the control actions across multiple systems to achieve a given set of high-level specifications.

A key requirement for the automatic task planning is in determining the appropriate abstraction models. A state-of-the-art problem is to determine the appropriate formalism to encode high-level specifications of tasks and the consequent methodologies of converting those into low-level action descriptions. Applications of formal methods to optimization problems provide abstractions for modeling the continuous agents' actions in an appropriate discrete domain. Task planning problems have a wide variety of applications, such as robotics, logistics, healthcare, traffic control, autonomous vehicles etc..

This dissertation introduces a novel methodology for automatic task planning of (possibly) heterogeneous multi-agent systems utilizing concepts from automata theory in an appropriate form for the proposed methodology. The modeling of system's capabilities, constraints and failure modes is presented. The resulting solution is guaranteed to be complete and optimal, while a heuristic approach is proposed for size reduction of the discrete domain of the system.

This dissertation includes a formal analysis leveraging the power and effectiveness of discrete abstractions models in solving complex task planning problems.

This dissertation provides industrial applications leveraging the proposed methodology to expound the modeling power of the proposed framework, while demonstrating its potential applications in providing solutions for the industry.

This dissertation introduces a new supervisory control framework for heterogeneous multi-agent control synthesis with reactive failure-mode based reconfiguration. Using concepts from automata theory in an appropriate form for the proposed methodology, this framework synthesizes the appropriate control actions for the multi-agent system to fulfill the given task specification with the capability of on-the-fly optimal reconfiguration of the task plan in case of failure occurrence.

Keywords: Task planning, Multi-Agent Systems, Discrete Event Systems, Discrete abstraction models, modeling, Automata theory, Supervisory Control, Automated planning, AI planning, Robotics, Artificial Intelligence.

Περίληψη

Ένας από τους κυριότερους τομείς ερευνητικού ενδιαφέροντος στον τομέα της ρομποτικής είναι η λειτουργία αυτόνομων συστημάτων πολλαπλών πρακτόρων για την εκτέλεση περίπλοκων εργασιών σε δυναμικά περιβάλλοντα. Επιπλέον, ο συνδυασμός σχεδιασμού διαδικασιών με εποπτικό έλεγχο για αυτόνομα συστήματα πολλαπλών πρακτόρων είναι πολύ απαιτητικός και δύσκολος, για αυτό και αποτελεί ένα θέμα αυξανόμενου ενδιαφέροντος την τελευταία δεκαετία, καθώς μπορεί να παρέχει ένα ισχυρό πλαίσιο για αυτόνομο σχεδιασμό και εκτέλεση διαδικασιών πολλαπλών πρακτόρων. Ενώ ο σχεδιασμός διαδικασιών λαμβάνει υπόψιν τον προσδιορισμό των διακριτών ενεργειών που εκτελούνται σε ακολουθία, διασφαλίζοντας με αυτό τον τρόπο ότι το σύστημα επιτυγχάνει τους στόχους που έχουν τεθεί, ο εποπτικός έλεγχος εξετάζει την ενορχήστρωση των ενεργειών ελέγχου σε πολλαπλά συστήματα για την επίτευξη ενός συνόλου διαδικασιών υψηλού επιπέδου.

Μια βασική απαίτηση για τον αυτόματο σχεδιασμό διαδικασιών είναι ο καθορισμός των κατάλληλων διακριτών μοντέλων αφαίρεσης. Ένα από τα κύρια προβλήματα της σύγχρονης τεχνολογίας είναι ο προσδιορισμός του κατάλληλου φορμαλισμού για την κωδικοποίηση των προδιαγραφών και των διαδικασιών σε υψηλό επίπεδο, αλλά και η μελέτη της μεθοδολογίας για την μετατροπή αυτών σε περιγραφές χαμηλού επιπέδου. Οι εφαρμογές τυπικών μεθόδων για διακριτά προβλήματα βελτιστοποίησης παρέχουν διακριτά μοντέλα αφαίρεσης, μοντελοποιώντας έτσι τις διαδικασίες των συνεχών πρακτόρων σε έναν διακριτό χώρο. Προβλήματα σχεδιασμού διαδικασιών συναντούμε για παράδειγμα στην ρομποτική, στην εφοδιαστική αλυσίδα, στην υγειονομική περίθαλψη, στον έλεγχο κυκλοφορίας οχημάτων, στα αυτόνομα οχήματα κ.α..

Αυτή η διατριβή εισάγει μια νέα μεθοδολογία για τον αυτόματο σχεδιασμό διαδικασιών συστημάτων πολλαπλών (πιθανώς) ετερογενών πρακτόρων, που χρησιμοποιούν έννοιες από τη θεωρία των αυτόματων, κατάλληλα διαμορφωμένες για την προτεινόμενη μεθοδολογία. Παρουσιάζεται η μοντελοποίηση των δυνατοτήτων, των περιορισμών και των τρόπων αποτυχίας του συστήματος. Η λύση που προκύπτει είναι εγγυημένα πλήρης και βέλτιστη, ενώ προτείνεται μια ευρετική προσέγγιση για τη μείωση του μεγέθους του διακριτού χώρου του συστήματος.

Στην παρούσα διατριβή παρουσιάζεται μια επίσημη ανάλυση που αξιοποιεί τη δύναμη και την αποτελεσματικότητα των διακριτών μοντέλων αφαίρεσης για την επίλυση σύνθετων προβλημάτων για τον αυτόματο σχεδιασμό διαδικασιών.

Ακόμη, η διατριβή αυτή παρέχει την περιγραφή μερικών βιομηχανικών εφαρμογών, αξιοποιώντας την προτεινόμενη μεθοδολογία, με σκοπό να παρουσιάσει τη δύναμη της μοντελοποίησης της προτεινόμενης μεθοδολογίας, ενώ καταδεικνύει τις πιθανές εφαρμογές αυτής στην παροχή λύσεων για αυτόματα συστήματα στον κλάδο της βιομηχανίας.

Τέλος, στην διατριβή αυτή εισάγεται ένα νέο πλαίσιο ελέγχου εποπτείας για την σύνθεση ελέγχου συστημάτων πολλαπλών ετερογενών παραγόντων με δυνατότητα επαναδιαμόρφωσης σε περιπτώσεις αποτυχίας του ελεγχόμενου συστήματος. Χρησιμοποιώντας έννοιες από τη θεωρία των αυτομάτων, κατάλληλα διαμορφωμένες για την προτεινόμενη μεθοδολογία, αυτό το πλαίσιο συνθέτει τις κατάλληλες διαδικασίες ελέγχου για το σύστημα πολλαπλών ετερογενών πρακτόρων για την εκπλήρωση των στόχων που έχουν τεθεί με τη δυνατότητα της βέλτιστης αναδιαμόρφωσης του σχεδίου διαδικασιών σε

περιπτώσεις αποτυχίας των ελεγχόμενων πρακτόρων.

Λέξεις Κλειδιά: Σχεδιασμός διαδικασιών, Συστήματα πολλαπλών πρακτόρων, Συστήματα διακριτών συμβάντων, Διακριτά μοντέλα αφαίρεσης, Αυτόματα, Εποπτεία και Έλεγχος, Αυτοματοποιημένος σχεδιασμός, Σχεδιασμός διαδικασιών Τεχνητής Νοημοσύνης, Ρομποτική, Τεχνητή Νοημοσύνη.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

AE	Annealer Engine
AGVs	Automated Guided Vehicles
AI	Artificial Intelligence
DA3	Digital Annealer
DES	Discrete Event Systems
DFAs	Deterministic Finite Automata
DIOA	Deterministic Input - Output Automata
DOF	Degree-of-Freedom
HIOA	Hybrid Input/Output (I/O) Automata
I/O	Input / Output
L4MS	Logistics for Manufacturing
LTL	Linear Temporal Logic
MAS	Multi-Agent Systems
MCP	Module Composition Problem
MIP	Mixed Integer Programming
NFAs	Nondeterministic Finite Automata
ϵ_0-NFAs	Non-Deterministic Finite Automata with ϵ-transitions
PDDL	Planning Domain Definition Language
QUBO	Quadratic Unconstrained Binary Optimization
SCT	Supervisory Control Theory
SPECTER	SuPErvisory Control Task plannER
TADA	Timed-Abstracting Deterministic Automaton
TAHA	Timed-Abstracting Hybrid Automaton

1 Introduction

Automated planning and scheduling are fundamental components of Artificial Intelligent (AI) dealing with the automatic synthesis of the plan to achieve a given objective, while enabling intelligent systems to make efficient decisions in complex and dynamic environments.

Planning aims to synthesize a plan consisting of a set of sequential discrete actions or tasks that must be performed to reach a given objective while optimizing overall performance measures. Moreover, automated task planners could determine the transformations to be applied in each given state. On the other hand, scheduling involves allocating the required resources and time to execute these actions/tasks. The combination of planning and scheduling enables AI systems to manage tasks effectively, optimize the available resources and adapt to alternative conditions.

Dating back to research in the 1960s and 1970s, the field of automated planning and scheduling incorporates research from operational research, logic-based planning, constraint satisfaction, symbolic AI, and machine learning. Nowadays, there is a wide range of planning applications including robotics, autonomous vehicles, reconfiguration, problem diagnosis, logistics, manufacturing, software testing, exploration etc.

In the context of this thesis, planning addresses the automation of a class of mechanical systems, called *autonomous systems*, that have sensing, actuation, and computation capabilities [1]. In the field of robotics, planning is leveraged in different ways in the sense of different actions with different objectives, such as *task planning* [2], *motion planning* [1, 3] and *path planning*.

An abstract representation of the classes of planning problems from discrete to continuous domain is illustrated in Fig. 1.1, where the task planning presents the sequential discrete (high-level) actions, motion planning involves finding control actions (i.e. collision-free trajectories) to execute such high-level actions and path planning plans the route for the robot to travel.

Task planning determines a finite set of actions that can be applied to a discrete set of states and aims to compute a discrete sequence of actions that results in a transition from a starting state to a given desired goal condition, called task. In task planning, the task (also known as global task) is usually decomposed into sub-tasks (also known as local tasks) that comprise the sequence of actions ensuring the satisfaction of the given task.

Motion planning is the capability of an autonomous mechanical robotic system to perform actions in order to accomplish tasks or achieve goal arrangements of physical objects by safely navigating in the real world while avoiding obstacles [3]. A classic motion planning problem is the Piano Mover's Problem. Given the model of the house where the piano is located and a piano object, the motion planning problem aims to determine the piano's movements to transport it from one room to another avoiding obstacles.

Path planning deals with the determination of an optimal path from the initial position to the target position according to constraints of the real workspace (i.e. safely avoiding obstacles). Planning algorithms use a heuristic function to estimate the cost of reaching the goal state from a given state, such as A* [4], Dijkstra [5], D* [6] and its successors Focused D* [7] and D* Lite [8], and heuristic search algorithms. Moreover, manipulation planning involves planning paths in high-dimensional space based on the degree-

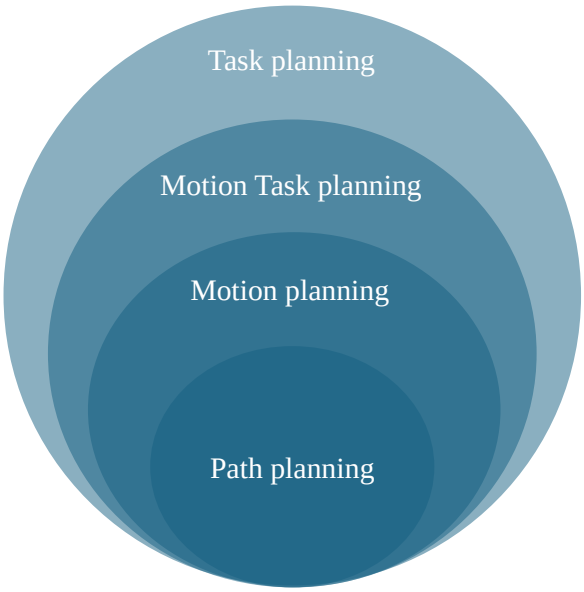


Figure 1.1: An abstract representation of the classes of planning.

of-freedom (DOF) and the kinematic constraints of the system.

Although task planning and motion planning share some similarities, they operate in different domains. Task planning refers to the agent's ability to determine sequences of complete actions in the workspace. On the other hand, motion planning refers to the agent's ability to safely navigate in the physical workspace. Task planning typically addresses the discrete domain while motion planning is carried out in the continuous domain. In the domain of motion task planning, a task plan is comprised of a sequence of motion plans.

Common terminology used in automated planning is:

- *State*, the current configuration or condition of the system expressed in a formal language.
- *Task*, the discrete action that should be performed so as to change the current state of the system.
- *Goal*, the desired state condition that the system should achieve.
- *Plan*, the sequence of actions that drive the system from an initial state to the desired state.
- *Planner*, a component that synthesizes the plan determining the resources and time required to achieve the goal.
- *Capability*, a requirement that enables transitions between the system's conditions.
- *Constraint*, a limitation that must be satisfied during the planning.
- *Failure*, a requirement that has not been achieved resulting the plan execution to be infeasible.

One of the main challenges in automated planning is scalability and complexity. The state space increases exponentially as the size of the problem grows, making it difficult to find solutions in a reasonable amount of time.

1.1 Task Planning for Multi-Agent Systems

Multi-Agent Systems (MAS) consist of autonomous entities known as agents. Agents use individual actions as well as interactions between other agents to solve tasks either independently or collaboratively. A multi-agent system consists of agents with a mix of discrete, continuous and hybrid dynamics interacting between themselves to achieve an objective. Multi-agent systems operate autonomously in dynamical environments to perform complicated tasks have been one of the major areas of research interest during the last decade.

Multi-agent task planning is the coordination of the tasks of multiple agents to optimize global objectives or achieve a common goal. High-level task planning using formal methods to define the system requirements is one of the promising approaches. Typical objectives arise from the multi-agent systems' behavior and requirements, such as sequential or reactive tasks, control, coordination and motion- and task- planning.

Several proposed methodologies address the high-level task planning problem using formal languages to express autonomous systems behavior and synthesize the supervisory control architecture to achieve the given specifications [9, 10]. Some of the most common approaches include Linear Temporal Logic (LTL), Capability Temporal Logic (CaTL) [11], Petri nets [12], sampling-based approaches [13, 14] and domain definition languages. Many of the existing works use LTL formulas to develop bottom-up [15–19] and distributed [20–22] approaches, where local LTL expressions are assigned to robots, or top-down approaches [23–27], where a global task is decomposed into independent sub-tasks that are treated separately by each agent. A descriptive overview of the literature using formal methods to accomplish high-level specifications can be found in [28].

In [15], a high-level plan is found by a discrete planner which seeks the set of system transitions that ensures the satisfaction of a logic formula. In [29], a formal method based on LTL has been developed to model specifications and implement centralized planning for multi-agent systems. In [30], the authors tackle the multi-robot task allocation problem under constraints defined by LTL formalism to concurrently plan tasks for robot agents. In [31], a formal method based on LTL has been deployed to model multi robot motion planning specifications. In [32] centralized controller synthesis for multi-agent systems from a global LTL specification is proposed where a globally connected multi-agent system is required in order to ensure information availability.

Some related works suggest that hierarchical abstraction techniques for single-agent systems can be extended to multi-agent systems using parallel compositions [24, 25]. However, the fact that the transitions should be on the common events to allow parallel execution restricts its application. In [17], temporal logic formulas are utilized to specify sub-formulas that could be executed in parallel by the agents, without a global task definition. In [12], a framework under Petri net formalism is proposed to decompose a global specification expressed in LTL formula in order to provide collision-free trajectories for a multi-robot system.

In [10], authors propose a formal synthesis of supervisory control software for multi-robot systems, while the scalability of the approach was improved in [9]. In [26], global specification is decomposable into local event sets to provide a distributed method from top to bottom to imply the accomplishment of the given specification. In [19], a framework for decentralized control of multi-agent systems from LTL spec-

ifications adopting a bottom-up formalism, assigning individual tasks to agents. The mutual satisfiability is verified through model checking techniques and it is not guaranteed beforehand.

There are several works addressing the (heterogeneous) multi-robot task allocation problem using LTL [30,31,33]. These approaches do not directly address the issue of inclusion of failure modes or on-the-fly re-planning. Other works tackle the task planning problem of heterogeneous multi-robot systems using CaTL [11, 34, 35], providing failure handling capabilities and on-the-fly re-planning. However partial agent failures (i.e. failure modes where an agent retains some functionality) are not directly addressed. Petri nets are a widely used modeling formalism for Discrete Event Systems (DES) [12, 36]. In [14], a sampling-based approach is proposed using directed trees to approximate the state space and transitions of synchronous product automata, providing probabilistic completeness and asymptotic optimality guarantees.

1.2 Task Planning in Manufacturing Logistics

Solving and automating complex manufacturing logistics tasks in order to produce a variety of products, has been one of the areas of research interest in the field of robotics and in particular their applications to manufacturing logistics. Shoe making is such a labor intensive process that requires many different sizes, styles and materials for shoes. The shoe manufacturing workflow undergoes various and complex steps before a quality product arrives at a customer, making it hard to automate the planning of such a production. Among the major challenges affecting shoe manufacturing are inefficient utilization of resources, including energy, materials and human resources, leading to increase the cost of the final product.

Manufacturing logistics workflows consist of multiple workflows with intermediate stages to produce a final product. To enhance the performance and reduce the production time, logistics robots are engaged to automate parts of the manufacturing workflow. Application of logistics robots entails the use of mobile automated guided vehicles (AGVs) in warehouses and storage facilities to organize and transport materials, products, items, goods etc. AGVs operate in predefined pathways transporting items in the factory. Another example of logistics robots is the robotic arms that grasp and sort items. AGVs are also utilized for goods transportation in agriculture, hospitals, medicine, laboratories, personal assistance etc.

1.3 Discrete Abstractions for Task Planning Problems

Automation and optimization of manufacturing logistics tasks has been one of the major areas of research interest during the last decades. In particular, discrete optimization problems have a wide variety of applications in the field. Of fundamental interest in the field of robotics is finding the appropriate formalism to encode high-level specifications of tasks and the consequent methodologies of converting those into low-level action descriptions. A key requirement for the automatic task planning is in determining the appropriate abstraction models.

Applications of formal methods to discrete optimization problems provide for discrete abstractions, modeling the continuous agents' actions in an appropriate discrete domain. Task planning is usually considered in the discrete domain where given an initial state and goal, the task planer aims to generate a

sequence of intermediate tasks to guide a team of agents (robots, workers, machines, etc.) to accomplish the goal. Automata are expressive abstractions of systems and they appear frequently in the literature in various forms [37–40]. Several research works address the challenges of task and motion task planning problems utilizing discrete abstractions and temporal logic [41–47].

Some of the state-of-art optimization tools are: the Gurobi optimizer [48], a Mixed Integer Programming (MIP) solver supporting quadratic objectives; the NVIDIA cuOpt [49] that employs GPU-accelerated logistics solvers relying on heuristics, metaheuristics and optimization with constraints; the Quantum-inspired annealers such as Fujitsu Digital Annealer (DA3) [50] for Quadratic Unconstrained Binary Optimization (QUBO) problems; and Fixstars Annealer Engine (AE) [51], a Cloud Platform for Quantum Annealing using GPUs; the OpenJij [52] that is a heuristic optimization library that offers solutions to QUBO and Ising models.

1.4 Control Synthesis of Multi-Agent Systems

During the last decade, one of the major areas of research interest in robotics has been autonomous operation of multi-agent systems in dynamical environments in order to perform complicated tasks. In particular, task planning and supervisory control for autonomous multi-agent systems have been topics of increasing interest since the combination of both can provide a robust framework for autonomous multi-agent task planning and execution. Task planning considers the determination of discrete sequences of actions that, when appropriately executed, are guaranteed to drive a system towards a desired outcome. Supervisory control refers to the orchestration of the control actions across multiple systems, so that the provided discrete actions will be appropriately executed in order to satisfy the given high-level specification.

The problem of controller synthesis from high-level specifications for multi-agent systems have been considered in several works. Many of them tackle the motion task specification and motion planning problems applying high-level task planning techniques to formalize the desired tasks and constraints. Formal languages, automata and temporal logic are some of the most common modeling and analysis techniques to formulate high-level abstractions of multi-agent systems. There are formal-methods based approaches utilized for system design purposes by proposing appropriate specification for system development [53, 54]. Moreover, formal verification [55] and controller synthesis are two of the most commonly utilized approaches to provide correctness guarantees with respect to formal requirements. The concept of automated controller synthesis from formal specifications is referred to as correct-by-construction controller synthesis scheme [56, 57], where given a control system along with the specification, the closed-loop control synthesis is orchestrated so that the specification is guaranteed to be satisfied.

Supervisory Control Theory (SCT) [58] is considered as the basic principle approach for controlling and monitoring of DES, where the behavior of the DES is restricted by a language modeling some specification in order to prevent the DES from forbidden behavior. The framework of supervisory control can be performed either in centralized or in decentralized settings [59] depending on each analysis approach. Several approaches have been developed for centralized [60–63] or decentralized [64–66] controller synthesis incorporating failure accommodation [67–72]. In [73], a prioritized planning approach is proposed

to synthesizes supervisors for solving a scheduling problem using SCT and a fragment of timed automata. In [9], an abstraction-based approach is proposed for synthesizing supervisory control strategies for multi-agent DES. A system modeling framework based on Hybrid Input/Output Automata (HIOA) was introduced by Lynch [40] for modeling non-deterministic state machines whose state may change instantaneously through a transition. In [74], a framework for fault-tolerant control of DES modeled as deterministic input/output (I/O) automata is introduced, where controller reconfiguration is implied after a fault occurrence. In [75], the concepts of feedforward and state-feedback control frameworks for deterministic plants are introduced whereas in [76] a feedback control framework for nondeterministic plants is presented. Utilizing the concept of deterministic I/O automata, the controllers can react to the outputs of the plant in order to steer the plant from its initial state into a desired final state.

On the other hand, LTL-based approaches have received significant attention. In [77], authors deal with independent LTL specifications and connectivity constraints for distributed motion and task control for multi-agent systems. In [78], a top-down approach is proposed for distributed control from global task specifications expressed as LTL formulas into local specifications. In [79], a correct by construction solution for synthesizing motion tasks based on LTL specifications with real-time objectives is introduced. In [80], a methodology for automatically synthesizing motion tasks based on LTL specifications is proposed.

1.5 Motivation

The motivation of this thesis comes from the field of Industry 5.0, where we would like to task autonomous multi-agent systems with high level task descriptions and have the system automatically synthesize the appropriate controllers to achieve reactive execution and fault accommodation.

Real world problems that can benefit from automatic task planning appear in multiple sectors, including logistics and supply chain management (e.g. vehicle routing, inventory management, warehouse operations), manufacturing and production (e.g. production scheduling, supply chain planning, maintenance scheduling, recycling planning), healthcare (e.g. patient scheduling, disaster response, drug development), transportation (e.g. air traffic control, public transportation, autonomous vehicle navigation), agriculture, project management etc.

Autonomous driving involves a number of planning algorithms and systems from global navigation to local trajectory tracking so as to determine the control inputs and motion task actions for a concrete driving route regarding vehicle kinematics and obstacles. Moreover, autonomous traffic control management is an automatic high-level control strategy that ensures a safe and valid plan for the vehicle traffic (e.g. cars, trains, airplanes etc.). Furthermore, automation in agriculture defines the farm management based on the AI technologies to deal with the spatial and temporal variability associated with all aspects of agricultural production. For example, the harvesting task is limited to pick one fruit, while the planning required for picking the rest is avoided. On the other hand, the project management incorporates many different types of planning, such as scheduling, monitoring and control.

Unfortunately, task planning in robotic domains is computationally challenging when the planning domain is large and complex [81]. Especially, in multi-objective planning problems, where complex global missions include multiple sub-goals. Thus, it is important to seek ways to reduce the complexity of the

problem's domain, such that a task plan can be found. Based on the abstraction level of the problem's domain, in a way that concentrates on a set of essential environmental characteristics, the problem complexity could be reduced so as to find a (optimal) solution in a timely manner.

1.6 Approach to solution

In this dissertation, *agents* are considered as the autonomous entities in the manufacturing workflow that are acting/reacting to their surroundings during a process or trigger event while being influenced by other agents. Agents could be robots, AGVs, humans, machines, items, goods, materials, products, work-cells, work stations etc.

An example of a task planning problem could be the following. Consider a factory floor with several robots, humans, machines, items, products, storage locations, intermediate processing areas, packing areas, buffer zones and so on. All these entities are considered as agents. An agent could have its own capabilities, constraints and some additional capabilities that emerge when the agent needs to cooperate with other agents to perform a specific action. Other agents may restrict the capabilities of an agent, and failures that may occur need to be considered. Lets assume that the goal is to produce a final product. This problem is at its core a task planning problem. To solve the task planning problem, we need to identify the sequence of actions that need to be performed by those agents in order to satisfy the task specification.

Following this concept, an approach to tackle the task planning problem is to start modeling what is happening in the factory floor in the discrete domain utilizing automata (Fig. 1.2). Assuming we have the appropriate abstractions of the agents' operations (i.e. modeled as automata) as well as of additional capabilities and constraints that emerge when multiple agents are present, working and interacting between themselves, the task planning problem is to find the sequence of actions that need to be performed by each agent to finally satisfy a specific goal, such as the manufacturing of a final product.

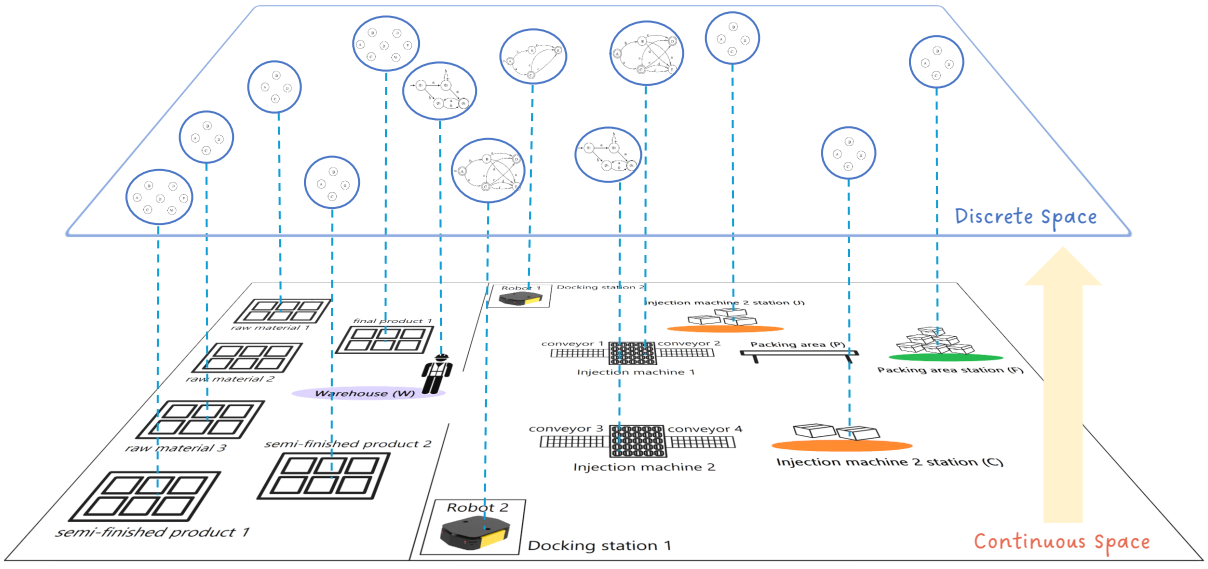


Figure 1.2: Discrete abstraction presentation of a factory plant.

Having defined the task specification and having the sequence of actions that satisfies the task specification produced by a task planner (i.e. the SPECTER task planner) in an appropriate form for our development, we need to orchestrate the control actions for each agent involved in the system in order to guarantee that the task specification will be achieved through the provided solution. Otherwise, in case of failures we need to automatically reconfigure the task plan until the task specification is satisfied. The abstracted supervisory control architecture of the factory plan presented in Fig. 1.2 is illustrated in Fig. 1.3.

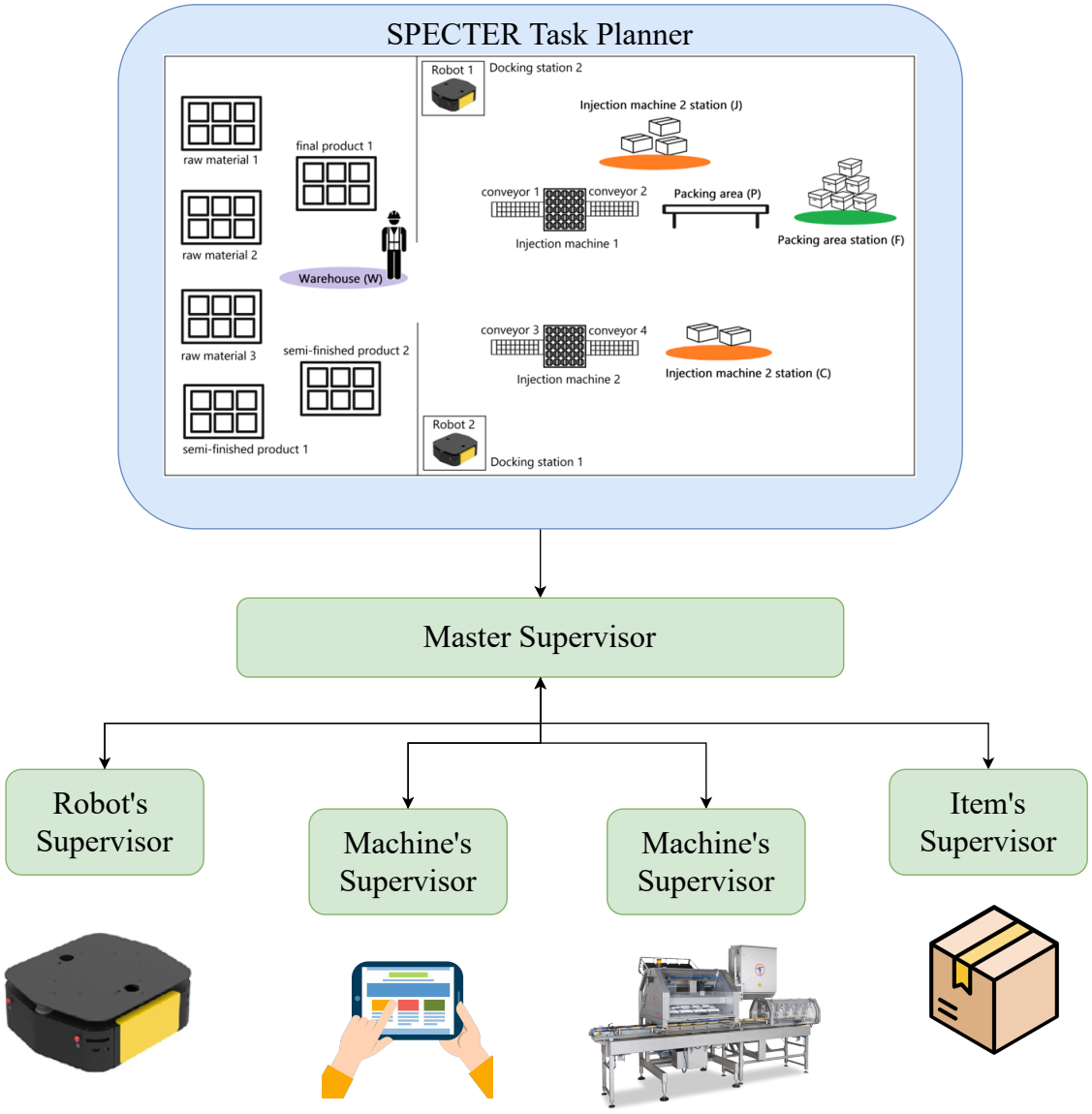


Figure 1.3: Supervisory control architecture of a factory plant.

1.7 Contributions of Dissertation

The main contributions of this dissertation is the analysis and the development of a new formal framework based on automata theory for modeling, analysis and control of the behavior of multi-agent systems. The key results of the analysis are presented as follows.

- A novel methodology is introduced for automatic task planning of (possibly) heterogeneous multi-agent systems. The formal framework is developed based on Nondeterministic Finite Automata (NFAs) with ϵ-transitions, where given the capabilities, constraints and failure modes of the agents involved, any initial state of the system and a task specification, an optimal solution is generated that satisfies the system constraints and the task specification. The resulting solution is guaranteed to be complete and optimal; moreover a heuristic solution that offers significant reduction of the computational requirements while relaxing the completeness and optimality requirements is proposed. The constructed system model is independent from the initial conditions and the task specifications, eliminating the need to repeat the costly pre-processing cycle, while allowing the incorporation of failure modes on-the-fly.
- The synthesis and reconfiguration (in case of failure occurrences) of the appropriate control actions is determined, so that the multi-agent system can satisfy the task specification as long as a solution exists.
- The SPECTER task planner is the software platform that has been developed to implement the proposed methodology. Using the abstraction model of the system in a language specifically developed for the planner, the SPECTER task planner yields optimal (or optionally sub-optimal results) to satisfy given objectives, based on the capabilities, constraints and failures modes of the system. The SPECTER task planner has been partially supported and implemented in several European Union's research and innovation programs under grant agreements 767642 (Logistics for Manufacturing (L4MS) / H2020), 951813 (Better Factory / H2020) and 101092295 (CIRCULOOS/Horizon Europe).
- A formal analysis leveraging the power and effectiveness of discrete abstractions models in solving complex task planning problems is provided.
- The effectiveness of the SPECTER task planner in various industrial cases is investigated to demonstrate the effectiveness and validity of the methodology. The case studies presented in this thesis are motivated be experiments implemented in European Union's projects from companies belonging to the L4MS and Better Factory consortia.
- An application in shoe manufacturing logistics leveraging the SPECTER task planner is presented to expound the modeling power of the proposed framework, while demonstrating its potential applications in providing solutions for the industry. An abstract model of the workflow of a shoe manufacturing company is developed taking into account the workflow stages, the agents involved in the production line and the temporal costs of each process. Based on the derived abstraction, several scenarios are studied to provide optimal solutions based on an appropriate chosen cost function.
- A new supervisory control framework for heterogeneous multi-agent control synthesis with reac-

tive failure-mode based reconfiguration is proposed. Leveraging the SPECTER framework, the proposed supervisory control framework synthesizes the appropriate control actions for the multi-agent system to satisfy the given task specification with the capability of on-the-fly optimal reconfiguration of the task plan in case of failure occurrence.

- The concept of time-abstracting automata is expanded to address agents with discrete and hybrid dynamics incorporating modeled and unmodeled failure modes, providing a connection to time-abstracting controllers.

Common among the works presented in the literature survey is the adoption of formal verification techniques for motion planning and controller synthesis. Several gaps in the existing literature are been addressed by our approach:

- First, to the best of our knowledge, none of the existing methodologies provides an effective modeling and execution framework capturing heterogeneous multi-agents with individual and collective capabilities, constraints and failure modes. Planning Domain Definition Language (PDDL) implementations [82–84] have exponential complexity and are not particularly suited to applications requiring on-line re-planning or reconfiguration.
- Second, in LTL-based planners, the solution domain depends on the task specification. Every time the specification is changed a new solution domain needs to be created, incurring increased computational overhead in the case of re-tasking (e.g., in the case of failures), modified objectives or initial conditions. In the current dissertation, a distinction is made between task specification and objective in the sense that the objective denotes the final state of a single agent while the task specification may encapsulate any number of objectives. Compared with the terminology used in the LTL literature, the evolution of the system is not part of the task specification but it is encoded in the system's capabilities and constraints.
- Third, increasing the objectives in a task specification, increases the computational complexity, particularly in LTL-based planners where the worst-case computational complexity is double exponential with respect to the length of the formula.
- Fourth, sampling-based approaches relax the properties of optimality and completeness in finite time. Some, like in [14], provide asymptotic guarantees as the computational resources asymptotically expand to capture the full state space.
- Fifth, PDDL is also used to describe planning problems solving the problem by finding the sequence of actions that reaches the goal state. However, the provided sequence of actions is static in the sense of not being reactive or adaptable to unforeseen events and failures.

Moreover, several approaches have been proposed for synthesizing control strategies for multi-agent systems utilizing formal languages or model checking techniques to satisfy formulas expressible in temporal logic. However, there are some gaps in the existing literature, that our approach aims to address. To the best of our knowledge, none of the existing methodologies provides a framework for multi-agent systems to handle the task plan reconfiguration in occurrence of failures that cause tasks to exceed their allocated temporal cost.

1.8 Structure of the Thesis

The organizational philosophy of this thesis is as follows:

- *Chapter 1* describes the background of the automatic task planning and supervisory control while presenting the related work, the motivation and the contribution of this dissertation.

  In *Part I*, *Chapter 2* proposes a formal framework for the automatic task planning problem for possibly heterogeneous multi-agent systems while introducing the necessary definitions and notations for the proposed methodology. The approach is validated through the presentation of case studies results.

- In *Part II*, *Chapter 3* presents an abstraction technique for multi-agent systems utilizing the task planning framework proposed in Chapter 2. The abstraction technique is used to model the system in an appropriate language for the SPECTER task planner and it is described with examples.

- In *Part II*, *Chapter 4* presents an application of SPECTER task planner for a shoe manufacturing logistics case study leveraging the proposed approach of Chapter 2.

- In *Part III*, *Chapter 5* proposes a new supervisory control framework for heterogeneous multi-agent control synthesis with reactive failure-mode based reconfiguration leveraging the task planner framework of Chapter 2; while introducing the necessary definitions and notations to support the proposed approach.

- In *Part IV*, *Chapter 6* concludes this dissertation presenting a summary of the thesis results and states the future research topics and extensions of the current thesis.

Part I

Automatic Task Planning

2 Automatic Task Planning for Multi-Agent Systems

This chapter introduces a new approach the SuPErvisory Control Task plannER (SPECTER) for high-level task planning problems with respect to agent capabilities, constraints and failure modes. The problem formulation uses the environment model composed by individual agents' capabilities and constraints, considering the individual agent's failure modes to determine the optimal task plan.

A new framework for high-level task planning problems with respect to agent capabilities, constraints and failure modes is introduced. This work builds on top of the work on motion task planning in [80], where module composition was used to automatically synthesize motion controllers defined by LTL specifications. While the current work generalizes the concept of motion tasks to general tasks and develops the formal machinery for their automatic synthesis, leveraging the agents' capabilities and constraints, it significantly departs from [80] by addressing a more general problem in a unified manner without the need to resort to the formalism of hybrid systems and LTL. The problem formulation uses the global system model as the environment model composed by individual agents' capabilities and constraints, considering the individual agent's failure modes to determine the optimal task plan. The problem is posed as a special case of module composition problem (MCP) [85] and then reduced to a combinatorial optimization problem of shortest directed path, which can be solved in polynomial time. The optimal module chain under a supervisory control framework enables the use of the generated module chain as a model system for building supervisory controllers, as demonstrated in [80]. In the sequel, the module chain with the supervisory controllers can e.g. be combined with Navigation Function based controllers [57] with time abstracting properties (i.e. guarantees on the required time for a task - see e.g. [86]), to synthesize control laws with performance guarantees that ensure compositionality for accomplishing individual and global tasks. The developed methodology provides the capability for on-the-fly incorporation of failure modes, however the supervisory control and navigation framework for reactive activation of failure modes during execution, and on-the-fly plan reconfiguration, are beyond the scope of the current work. The active source code of the software developed is available at [87].

The main contributions of this chapter are:

- A new system modeling framework that combines the agents' capabilities and constraints both at the individual and the group levels, including failure modes.
- Determination of optimal task plans that satisfy any possible task specification from any initial condition, without the need to repeat the pre-processing step.
- Capability of incorporation of multiple objectives in the task specification, decreasing the time complexity while maintaining the space complexity.
- Determination of reduced complexity sub-optimal solutions for any task specification, while maintaining its space complexity.
- Capability of incorporation of failure modes on-the-fly for re-planning, without the need to repeat the costly pre-processing step.

The Chapter is organized as follows: Section 2.1 presents the necessary preliminary notions, Section 2.2 presents the problem formulation using a new framework of the Epsilon Non-Deterministic Finite Automata (ϵ-NFAs) formalism while Section 2.3 exploits the ϵ_0-NFA formalism to reduce the problem to a Module Composition one. Section 2.4 presents the algorithms and the analysis of the methodology while Section 2.5 presents two case studies to demonstrate the effectiveness of the proposed framework.

2.1 Preliminaries

2.1.1 Definitions

If A and B are sets, the cardinality of set A is denoted as $|A|$. The union and intersection of sets are denoted as $A \cup B$ and $A \cap B$ whereas set subtraction of set B from set A is denoted as $A \setminus B$. Operator $\wedge$ denotes conjunction.

We will use the definition of Deterministic Finite Automata (DFAs) by [88].

Definition 1 (DFA)**.** *A Deterministic Finite Automaton (DFA) is a six-tuple,* $G = (X_G, E_G, f_G, \Gamma_G, x_{0,G}, X_{m,G})$*, consisting of:*

- *a non-empty finite set of states* X_G*,*
- *a non-empty finite set of events* E_G*,*
- *a transition function* $f_G : X_G \times E_G \to X_G$*,*
- *an active event function* $\Gamma_G : X_G \to 2^{E_G}$*,*
- *an initial state* $x_{0,G} \in X_G$*,*
- *a finite set of marked states* $X_{m,G} \subseteq X_G$

where f_G *is partial on its domain in the sense that if event* $e \in \Gamma_G(x)$*, then* e *labels a transition from* x *to a unique state* $y = f_G(x, e)$*.*

Unless otherwise stated, subscripts of an (DFA or NFA) automaton's tuple elements will refer to the corresponding automaton as above. An example of DFA is shown in Fig. 2.1.

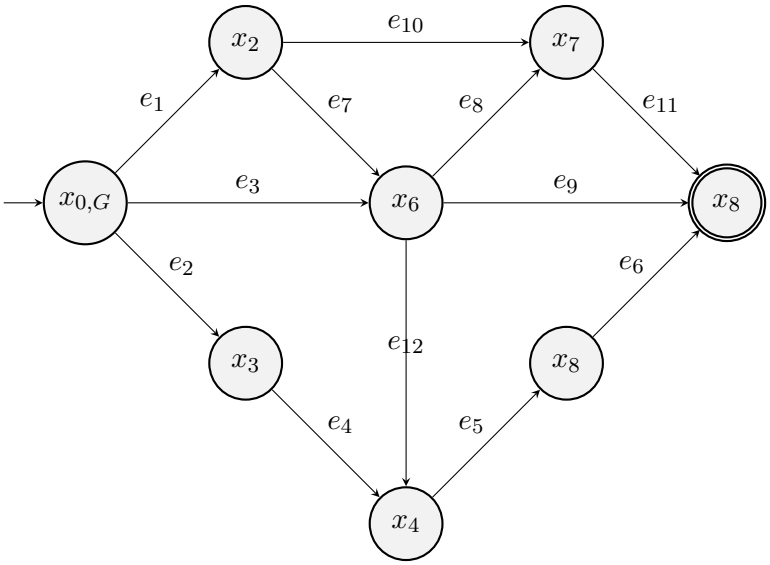


Figure 2.1: DFA G.

We need to introduce a new framework of ϵ-NFAs that allows the use of ϵ-transitions by enabling transitions on an empty event only from the initial state. The introduction of these automata became necessary to enable closure under the newly introduced operations in Section 2.1.2. In practical terms, this allows us to consistently setup our system to handle any initial condition.

Definition 2 (ϵ_0-NFA)**.** *An ϵ_0-NFA is a Nondeterministic Finite Automaton with ϵ-transitions only from the initial state, defined as a six-tuple:*

$$^{\epsilon_0}G \triangleq (X_G \cup \{x_0\}, E_G \cup \{\epsilon\}, {}^{\epsilon_0}f_G, \Gamma_G, x_0, X_{m,G}),$$

where:

$$^{\epsilon_0}f_G : X_G \cup \{x_0\} \times E_G \cup \{\epsilon\} \to 2^{X_G}$$

such that:

$$\forall (x, e) \in X_G \times E_G : {}^{\epsilon_0}f_G(x, e) := f_G(x, e)$$

where $x_0 \notin X_G$ is the initial state with $x_0 \neq x_{0,G}$.

These components have the same interpretation as in the Definition 1, with the following differences:

1. The initial state x_0 is not an element of X_G.
2. The event ϵ is not an element of E_G.
3. $^{\epsilon_0}f_G$ is a function partial on its domain.

Using the DFA G, the $^{\epsilon_0}G$ is constructed as shown in Fig. 2.2. In this case, the "virtual" starting state x_0 with an epsilon event ϵ [89] are added to the starting state $x_{0,G}$ of the DFA G.

Corollary 1. *If $^{\epsilon_0}G$ is an ϵ_0-NFA then it can be converted to the DFA G by removing x_0 along with the associated ϵ transitions and assigning an $x_{0,G} \in X_G$ as an initial state.*

Proof. This can be trivially shown by observing that the six-tuple obtained by the operation is as the one in the definition of DFA (Definition 1). □

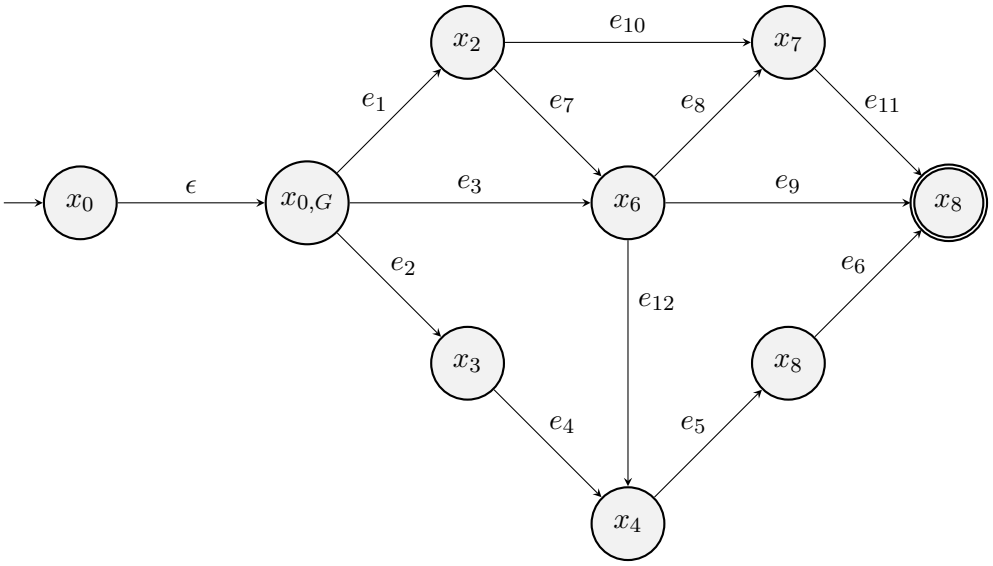


Figure 2.2: ϵ_0-NFA G.

Based on the above Corollary, we can construct the following operator:

Definition 3 (ϵ_0-NFA Determinization)**.** *Let ${}^{\epsilon_0}G$ be an ϵ_0-NFA and $x_{0,G} \in X_G$ a state in X_G that we want to assign as an initial state. Then*

$$\Delta({}^{\epsilon_0}G, x_{0,G}) \triangleq G$$

where G is the DFA and the determinization operator Δ performs the conversion as described in Corollary 1.

In this case, the determinization operator converts the ${}^{\epsilon_0}G$ to the DFA G by removing the "virtual" starting state x_0 and the ϵ event, and assigning an element of X_G as the new starting state $x_{0,G}$ of the constructed DFA G. Compared with the standard determinization in automata theory, the determinization operator in our case, results in a state space equivalent to the state space of ϵ_0-NFA, while the accommodation of ϵ transitions in the resulting DFA is not required.

We need to associate a cost function with each transition:

Definition 4 (Transition Cost Function)**.** *A transition cost function for an event set E_G, is defined as*

$$g_G : E_G \to \mathbb{R}_{>0}.$$

We define the inverse transition function that denotes the reverse transition relation that is derived from the transition function.

Definition 5 (Inverse Transition Function)**.** *Let G be a DFA. Define*

$$f_G^{-1} : X_G \times E_G \to X_G$$

to be the inverse transition function such that

$$f_G(x_G, e) = y_G \wedge f_G^{-1}(y_G, e) = x_G$$

for some

$$x_G : e \in \Gamma_G(x_G),\ y_G : e \in \Gamma_G(f_G^{-1}(y_G, e)).$$

Note that an inverse transition function as in Definition 5 cannot always be defined for DFAs. For DFAs where such inverse transition function can be defined, we introduce the following concept:

Definition 6 (Compatible ϵ_0-NFAs)**.** *Let ${}^{\epsilon_0}G$ and ${}^{\epsilon_0}B$ be ϵ_0-NFAs. Let*

$$E_C := E_G \cap E_B.$$

Then, the automata ${}^{\epsilon_0}G$ and ${}^{\epsilon_0}B$ are compatible iff $\forall e \in E_C$, it holds that[1]*,*

$$f_G(x_G, e) = f_B(x_B, e)$$

for some

$$x_G : e \in \Gamma_G(x_G),\ x_B : e \in \Gamma_B(x_B)$$

and

$$f_G^{-1}(y_G, e) = f_B^{-1}(y_B, e)$$

for some

$$y_G : e \in \Gamma_G(f_G^{-1}(y_G, e)),\ y_B : e \in \Gamma_B(f_B^{-1}(y_B, e)).$$

In such case it will also be true that:

$$x_G = x_B \text{ and } \mathrm{y_G} = \mathrm{y_B}.$$

We denote such a compatibility relation as:

$${}^{\epsilon_0}G \asymp {}^{\epsilon_0}B.$$

Note that the above definition effectively forces the endpoints of common events to be common states. Additionally, note that in the case where $E_C = \emptyset$, the compatibility requirement is automatically satisfied. Having the ${}^{\epsilon_0}B$ as shown in Fig. 2.3, the requirement of compatibility relation between ${}^{\epsilon_0}G$ and ${}^{\epsilon_0}B$ ensures that the transition on common events should have the same starting and the same ending state. In this case, event $e_4 \in \{E_G \cup E_B\}$ while it holds that $f_G(x_3, e_4) \equiv f_B(x_3, e_4)$ and $f_G^{-1}(x_4, e_4) = f_B^{-1}(x_4, e_4)$. Hence, ${}^{\epsilon_0}G \asymp {}^{\epsilon_0}B$.

2.1.2 Operations on compatible automata

Here we introduce a custom set of the basic operations on compatible ϵ_0-NFAs: union, subtraction and concatenation. The introduced operations differ from the ones found in the automata literature [88–90]

[1] Note that due to Corollary 1 we have that ${}^{\epsilon_0}f_G(x_G, e) \equiv f_G(x_G, e)$ and ${}^{\epsilon_0}f_B(x_B, e) \equiv f_B(x_B, e)$.

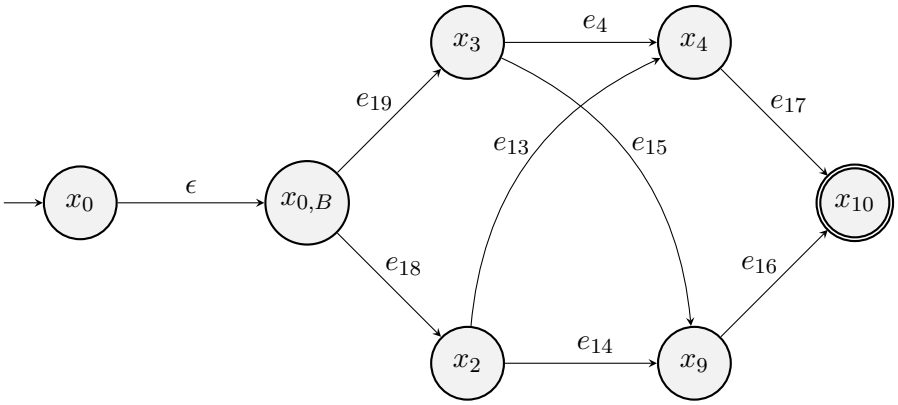


Figure 2.3: ϵ_0-NFA B.

and provide the necessary functionality for the subsequent developments.

Definition 7 (Union of Compatible Automata)**.** *For the ϵ_0-NFAs ${}^{\epsilon_0}G$ and ${}^{\epsilon_0}B$, assume ${}^{\epsilon_0}G \asymp {}^{\epsilon_0}B$. Define the compatible ϵ_0-NFAs union:*

$$ {}^{\epsilon_0}\mathcal{P} \triangleq {}^{\epsilon_0}G \cup_{\asymp} {}^{\epsilon_0}B $$

to be the six-tuple:

$$ {}^{\epsilon_0}\mathcal{P} \triangleq (X_{\mathcal{P}} \cup \{x_0\}, E_{\mathcal{P}} \cup \{\epsilon\}, {}^{\epsilon_0}f_{\mathcal{P}}, \Gamma_{\mathcal{P}}, x_0, X_{m,\mathcal{P}}), $$

where

$$ X_{\mathcal{P}} := X_G \cup X_B, $$

$$ E_{\mathcal{P}} := E_G \cup E_B, $$

$$ \Gamma_{\mathcal{P}} : X_{\mathcal{P}} \to 2^{E_{\mathcal{P}}}, $$

$$ x_0 \notin X_{\mathcal{P}} $$

and

$$ X_{m,\mathcal{P}} := X_{m,G} \cup X_{m,B}. $$

The transition function:

$$ {}^{\epsilon_0}f_{\mathcal{P}} : X_{\mathcal{P}} \cup \{x_0\} \times E_{\mathcal{P}} \cup \{\epsilon\} \to 2^{X_{\mathcal{P}}} $$

is such that:

$$ \forall (x, e) \in X_G \times E_G : {}^{\epsilon_0}f_{\mathcal{P}}(x, e) := f_G(x, e) $$

and

$$ \forall (x, e) \in X_B \times E_B : {}^{\epsilon_0}f_{\mathcal{P}}(x, e) := f_B(x, e). $$

Compared with the standard union operation on automata presented in the literature, the union operation of Definition 7 produces the union instead of the Cartesian product state space.

By implementing the union operation of Definition 7 on the compatible ϵ_0-NFAs ${}^{\epsilon_0}G$ (Fig. 2.2) and ${}^{\epsilon_0}B$ (Fig. 2.3), the new ϵ_0-NFA ${}^{\epsilon_0}\mathcal{P}$ is constructed as shown in Fig. 2.4.

Definition 8 (Subtraction of Compatible Automata)**.** *For the ϵ_0-NFAs ${}^{\epsilon_0}G$ and ${}^{\epsilon_0}B$, assume ${}^{\epsilon_0}G \asymp {}^{\epsilon_0}B$.*

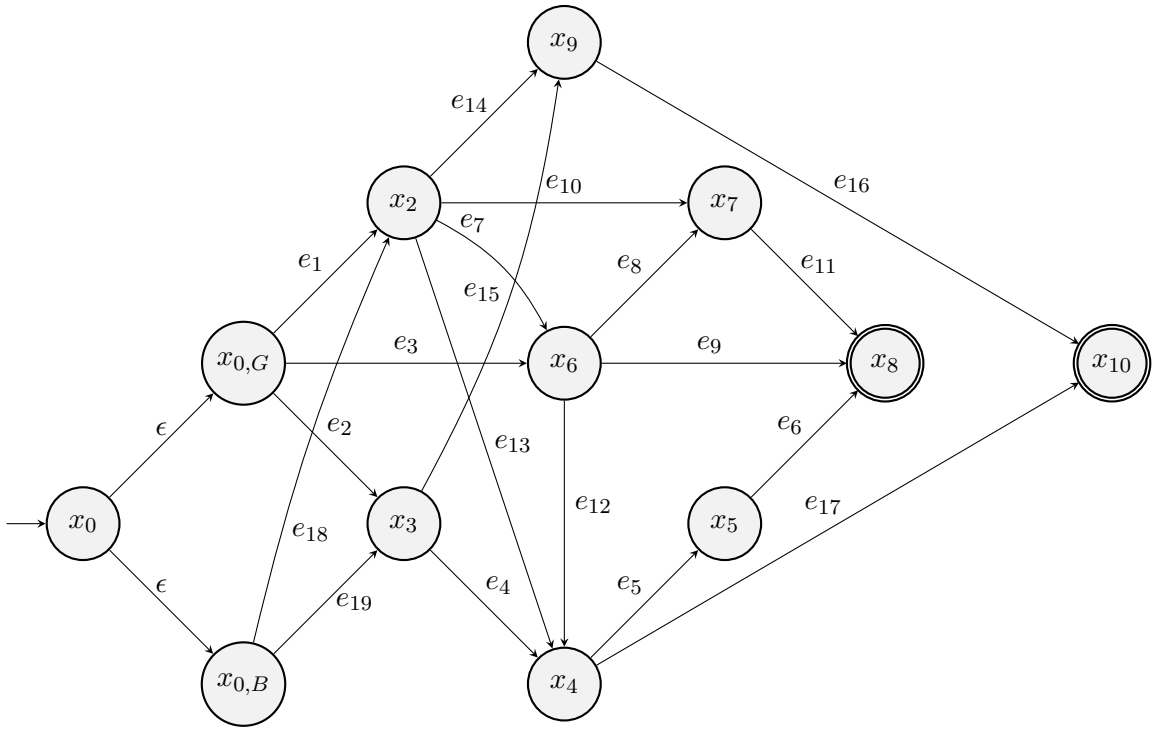


Figure 2.4: ϵ_0-NFA $\mathcal{P}$ constructed by implementing the union operation of Definition 7 on the compatible ${}^{\epsilon_0}G$ and ${}^{\epsilon_0}B$.

Define the compatible ϵ_0-NFAs subtraction:

$$ {}^{\epsilon_0}\Theta \triangleq {}^{\epsilon_0}G \setminus_{\asymp} {}^{\epsilon_0}B $$

to be the six-tuple:

$$ {}^{\epsilon_0}\Theta \triangleq (X_\Theta \cup \{x_0\}, E_\Theta \cup \{\epsilon\}, {}^{\epsilon_0}f_\Theta, \Gamma_\Theta, x_0, X_{m,\Theta}), $$

where

$$ X_\Theta := X_G, $$

$$ E_\Theta := E_G \setminus E_B, $$

$$ \Gamma_\Theta : X_\Theta \to 2^{E_\Theta} $$

and

$$ x_0 \notin X_\Theta, $$

$$ X_{m,\Theta} := X_{m,G} \setminus X_{m,B}. $$

The transition function:

$$ {}^{\epsilon_0}f_\Theta : X_\Theta \cup \{x_0\} \times E_\Theta \cup \{\epsilon\} \to 2^{X_\Theta} $$

is such that:

$$ \forall (x, e) \in X_\Theta \times E_\Theta : {}^{\epsilon_0}f_\Theta(x, e) := f_\Theta(x, e). $$

Note that the subtraction operation of Definition 8 is more in line with the set difference operator than

with the intersection with the language complement that is used in regular languages.

By implementing the subtraction operation of Definition 8 on the compatible ϵ_0-NFAs ${}^{\epsilon_0}G$ (Fig. 2.2) and ${}^{\epsilon_0}B$ (Fig. 2.3), the new ϵ_0-NFA ${}^{\epsilon_0}\Theta$ is constructed as shown in Fig. 2.5.

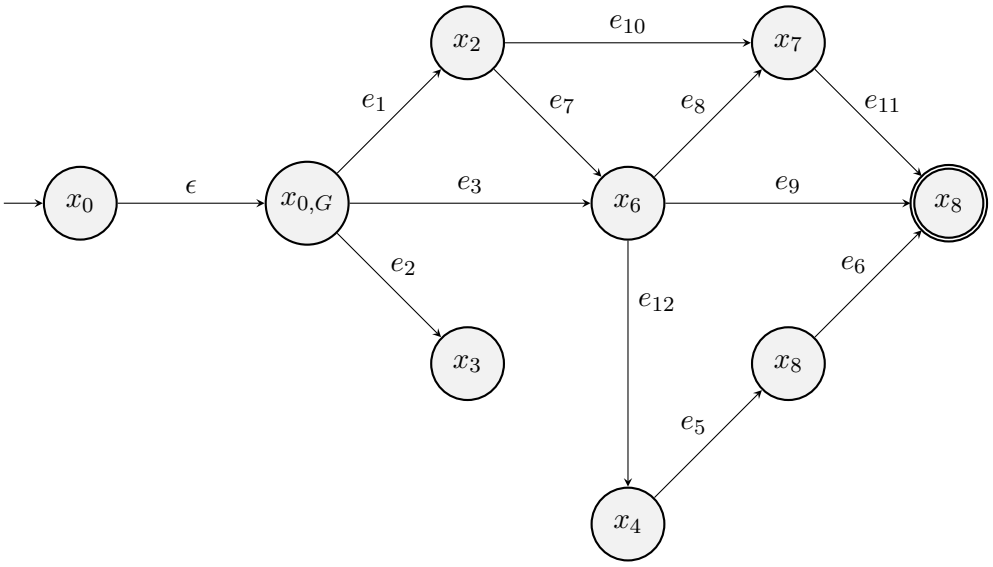


Figure 2.5: ϵ_0-NFA Θ constructed by implementing the subtraction operation of Definition 8 on the compatible ${}^{\epsilon_0}G$ and ${}^{\epsilon_0}B$.

Definition 9 (Concatenation of Compatible Automata)**.** *For the ϵ_0-NFAs ${}^{\epsilon_0}C$ and ${}^{\epsilon_0}Z$, assume ${}^{\epsilon_0}C \asymp {}^{\epsilon_0}Z$. Define the compatible ϵ_0-NFAs concatenation:*

$$ {}^{\epsilon_0}\Phi \triangleq {}^{\epsilon_0}C \amalg_{\asymp} {}^{\epsilon_0}Z $$

to be the six-tuple:

$$ {}^{\epsilon_0}\Phi \triangleq (X_\Phi \cup \{x_0\}, E_\Phi \cup \{\epsilon\}, {}^{\epsilon_0}f_\Phi, \Gamma_\Phi, x_0, X_{m,\Phi}), $$

where

$$ X_\Phi := \{uv | u \in X_C, v \in X_Z\}, $$

$$ E_\Phi := E_C \cup E_Z, $$

$$ \Gamma_\Phi : X_\Phi \to 2^{E_\Phi}, $$

$$ x_0 \notin X_\Phi, $$

and

$$ X_{m,\Phi} := \{uv | u \in X_{m,C} \wedge v \in X_{m,Z}\}. $$

The transition function:

$$ {}^{\epsilon_0}f_\Phi : X_\Phi \cup \{x_0\} \times E_\Phi \cup \{\epsilon\} \to 2^{X_\Phi} $$

is such that:

$$ \forall (x, e) \in X_\Phi \times E_\Phi : {}^{\epsilon_0}f_\Phi(x, e) := f_\Phi(x, e). $$

Note that the operation of concatenation in Definition 9 above, is similar to the well known parallel composition operation (see e.g. [88]) but differs in that the event sets undergo a disjoint union in our case to ensure that only one event can be activated at a time (no synchronization on common events needed).

Assuming we have the ϵ_0-NFAs C and Z as shown in Fig. 2.6 and Fig. 2.7, respectively. By implementing the concatenation operation of Definition 8 on the compatible ϵ_0-NFAs ${}^{\epsilon_0}C$ and ${}^{\epsilon_0}Z$, the new ϵ_0-NFA ${}^{\epsilon_0}\Phi$ is constructed as shown in Fig. 2.8.

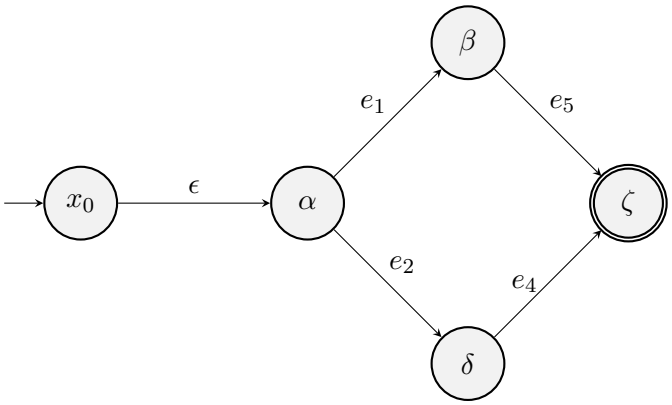


Figure 2.6: ϵ_0-NFA C.

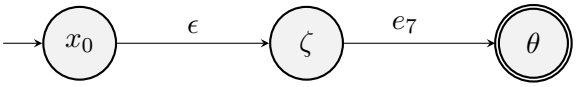


Figure 2.7: ϵ_0-NFA Z.

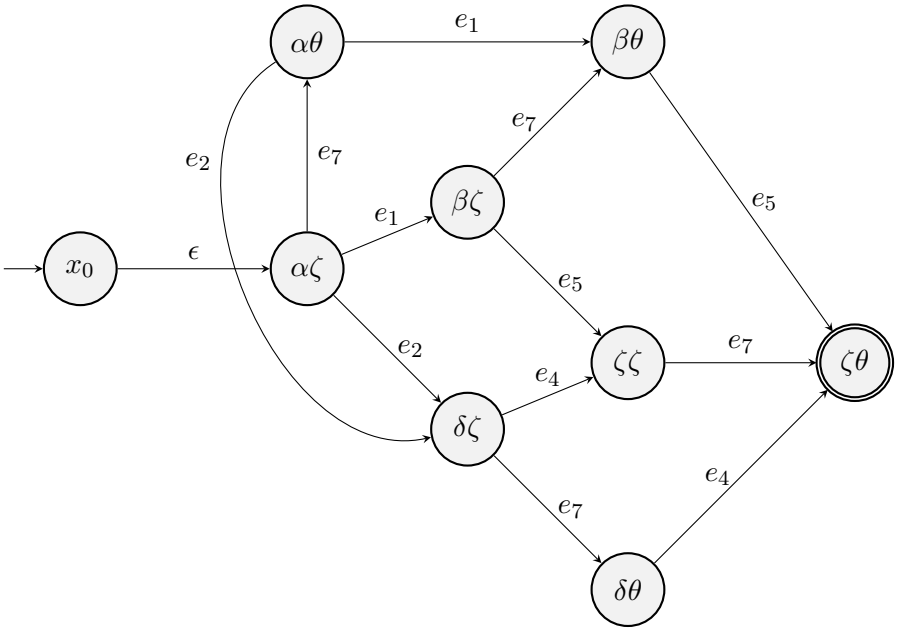


Figure 2.8: ϵ_0-NFA Φ constructed by implementing the concatenation operation of Definition 9 on the compatible ${}^{\epsilon_0}C$ and ${}^{\epsilon_0}Z$.

The concept of compatible automata, allows us to recover the following properties for the operations of Definition 7, Definition 8 and Definition 9:

Corollary 2. *The automata resulting from the operations defined in Definition 7, Definition 8 and Definition 9 are ϵ_0-NFAs.*

Proof. This is established by first noting that for the operations in Definition 7, Definition 8 and Definition 9, the resulting automata are six-tuples as in Definition 2. Now we need to show that the resulting transition function f_R is partial in its domain in the sense that if event $e \in \Gamma_R(x), x \in X_R$, then e labels a transition from x to a unique state $y = f_R(x, e)$. Assume that R is the resulting automaton from the operation (union, subtraction or concatenation). We have the following cases:

1. *Union operation:* Due to the properties of Definition 6, it will hold that, and $\forall x \in X_R$, then:

$$\forall e \in \Gamma_G(x) \cap \Gamma_B(x)$$

it will hold that:

$$f_G(x, e) = f_B(x, e),$$

2. *Subtraction operation:* Due to the properties of Definition 6, it will hold that, and $\forall x \in X_R$, then:

$$\forall e \in \Gamma_R(x)$$

it will hold that:

$$f_R(x, e) = f_G(x, e),$$

3. *Concatenation operation:* Due to the properties of Definition 6, it holds that, $\forall(x := uv) \in X_R$, where $u \in X_C$ and $v \in X_Z$, then:

$$\forall e \in \Gamma_R(x)$$

it holds that: either

$$f_R(x, e) = f_C(u, e), \text{ with } e \in E_C,$$

or

$$f_R(x, e) = f_Z(v, e), \text{ with } e \in E_Z.$$

As can be seen, in all three operations, $\forall x \in X_R$ and for each $e \in \Gamma_R(x)$, the state $f_R(x, e)$ is a unique state in X_R and hence, the resulting transition function is partial in its domain. Thus, R will be an ϵ_0-NFA since it fulfills the requirements of Definition 2. □

We need to introduce the following operator that addresses individual states of concatenated states of Definition 9.

Definition 10 (Projection Operator)**.** *Let $x \in X_\Phi$ be a concatenated state of n elements, where x_i denotes the $i'th$ element of x, $i \in \{1, \ldots, n\}$. Define the projector b as an n-bit binary and b_i its i'th bit. The projection operator is defined as the ordered set:*

$$proj(x, b) \triangleq \{x_i | b_i = 1\}.$$

We will need some definitions from Module Composition Theory literature as well as some new ones for our development. Using the module definition by [85], we state the module in an appropriate form for our development.

Definition 11 (Single Port Module)**.** *Let G be a DFA. Consider the finite set of ports $P = P_{in} \cup P_{out}$, where P_{in} is a non-empty finite set of input ports and P_{out} is a non-empty finite set of output ports with $P \subseteq X_G$ and $P_{in} \cap P_{out} \neq \emptyset$. We define the single input/single output port (I/O) module T as a 3-tuple of:*

$$T \triangleq \{p, e, q\},$$

where $p \in P_{in}$ is the input port,

$$e \in E_G : f_G(p, e) = q$$

and $q \in P_{out}$ is the output port.

An example of a single port module is shown in Fig. 2.9. The cost associated with the module T is defined as:

$$c(T) \triangleq g_G(e).$$

We can also define the inverted module T^{-1}, shown in Fig. 2.10, where its input becomes its output and vice versa such that Definition 5 holds.

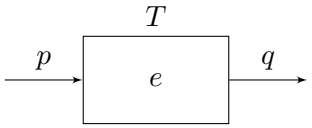


Figure 2.9: The single port module T.

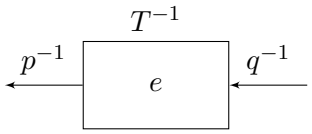


Figure 2.10: The inverted single port module T^{-1}.

We consider the notion of directional compatibility of single port modules, as defined in Definition 12, that specifies the interaction between the modules.

Definition 12 (Directional Compatibility of I/O Modules)**.** *Let G be a DFA and $T_\chi = \{p_\chi, e_\chi, q_\chi\}$ and $T_\psi = \{p_\psi, e_\psi, q_\psi\}$ be modules. We define the directional compatibility relation between T_χ and T_ψ and we denote this relation as:*

$$T_\chi \rightharpoonup T_\psi$$

iff $q_\chi = p_\psi$.

The connectivity of two compatible modules is shown in Fig. 2.11.

Figure 2.11: The connectivity of compatible modules T_χ and T_ψ.

2.2 Problem Formulation

Before proceeding with the problem formulation, let us formally state the problem in consideration.

2.2.1 Problem statement

Given a set of agents, their capabilities and constraints as individuals, their emerging capabilities and constraints when operating in teams, their failure modes, and a task specification defining the final state of at least one agent, determine (if any) the optimal sequence of actions that brings the system from any initial state, to a state satisfying the task specification.

Having formally defined the problem statement, and having the framework of ϵ_0-NFAs, we can model all possible behaviors of the system using appropriate abstractions. Considering that everything in the system can be expressed as ϵ_0-NFAs, we can model the behavior of the system using the operations on compatible automata. More specifically, capabilities and constraints can be combined using the union operation, system's capabilities can be disabled using the subtraction operation and capabilities and con-

straints of multiple agents can be combined using the concatenation operation. Following this procedure, the operational environment of the system is constructed.

2.2.2 Agent Model

Agents are considered as autonomous entities, such as humans, robots, items, machines or anything that could change its state to act on or react to its surroundings during a process or trigger event. The framework of ϵ_0-NFA is utilized to model the behavior of each agent in the system. An agent is modeled as an ϵ_0-NFA (see Definition 2) composed by the agent's capabilities and constraints. Agent's capabilities represent the available state transitions derived from the combination of individual capabilities and failure mode, expressed as ϵ_0-NFAs. Agent's constraints express the non-available state transitions derived from the union of individual's constraints that are modeled as ϵ_0-NFAs. The agent model is produced by the subtraction of the agent's constraints from the agent's capabilities. The i^{th} agent, $i \in \{1, \ldots, n\}$, is modeled by the ϵ_0-NFA ${}^{\epsilon_0}A_i$, and let ${}^{\epsilon_0}\mathcal{A}$ denote the set of n agents.

2.2.2.1 Individual Agent Capabilities

Let κ be the total number of individual capabilities of agent ${}^{\epsilon_0}A_i$ and let the ϵ_0-NFA ${}^{\epsilon_0}M_{\beta,i}$ denotes its β'th individual capability, where $\beta \in \{1, \ldots, \kappa\}$.

2.2.2.2 Individual Agent Failure Mode

Consider a state $x' \in X_{M_{\beta,i}}$. We define a failure mode of ${}^{\epsilon_0}A_i$ as the inability of an agent to complete the transition from some $x \in X_{M_{\beta,i}}$ to x' with an occurrence of event $e \in E_{M_{\beta,i}}$. This describes a detected transition failure of ${}^{\epsilon_0}A_i$ which renders $f_{M_{\beta,i}}(x, e) = x'$ infeasible. This failure mode is modeled by the ϵ_0-NFA ${}^{\epsilon_0}F_i$ such that $X_{F_i} = \{x, x'\}$ and $E_{F_i} = \{e\}$.

We model the agent's capabilities as the subtraction of the agent failure from the union of individual agent capabilities utilizing the union and the subtraction operations of compatible automata.

2.2.2.3 Agent Capabilities

Considering κ compatible individual agent capabilities such that ${}^{\epsilon_0}M_{\alpha,i} \asymp {}^{\epsilon_0}M_{\beta,i}$, $\alpha \neq \beta$ with $\alpha, \beta \in \{1, \ldots, \kappa\}$ and ${}^{\epsilon_0}F_i \asymp {}^{\epsilon_0}M_{\beta,i}$, the capabilities of ${}^{\epsilon_0}A_i$ are modeled by the ${}^{\epsilon_0}K_i$ as:

$$ {}^{\epsilon_0}K_i \triangleq \left\{ \bigcup_{\beta=1}^{\kappa} \asymp {}^{\epsilon_0}M_{\beta,i} \right\} \backslash \asymp {}^{\epsilon_0}F_i. \tag{2.1} $$

2.2.2.4 Individual Agent Constraints

Let λ be the total number of individual constraints of agent ${}^{\epsilon_0}A_i$ and let the ϵ_0-NFA ${}^{\epsilon_0}N_{\xi,i}$ denotes its ξ'th individual constraint, where $\xi \in \{1, \ldots, \lambda\}$.

2.2.2.5 Agent Constraints

We model the agent's constraints as the union of individual agent constraints utilizing the union of compatible automata operation. Considering λ compatible individual agent constraints such that ${}^{\epsilon_0}N_{\xi,i} \asymp {}^{\epsilon_0}N_{\eta,i}$, $\xi \neq \eta$ with $\eta \in \{1, \ldots, \lambda\}$, the constraints of ${}^{\epsilon_0}A_i$ are modeled by the ${}^{\epsilon_0}D_i$ as:

$$ {}^{\epsilon_0}D_i \triangleq \left\{ \bigcup_{\xi=1}^{\lambda} \asymp {}^{\epsilon_0}N_{\xi,i} \right\}. \tag{2.2} $$

Considering ${}^{\epsilon_0}K_i \asymp {}^{\epsilon_0}D_i$, the i^{th} agent is modeled by the ϵ_0-NFA ${}^{\epsilon_0}A_i$ after subtracting the ϵ_0-NFA ${}^{\epsilon_0}D_i$ from the ϵ_0-NFA${}^{\epsilon_0}K_i$:

$$ {}^{\epsilon_0}A_i \triangleq {}^{\epsilon_0}K_i \setminus_{\asymp} {}^{\epsilon_0}D_i. \tag{2.3} $$

2.2.3 Environment Model

Agents are acting in the environment to reach individual states while being influenced by state capabilities and constraints relating to other agents. Agents who rely on other agents to perform actions or reach goals are grouped into a team. The capabilities and constraints of the team are modeled as a combination of individual states of the members of the team. These capabilities and constraints, called inter-agent capabilities and constraints, express respectively the available and non-available environment state transitions of those agents. Given the agents' capabilities and constraints and the inter-agent capabilities and constraints, the environment model captures all the possible combinations of agents' states. The procedure of constructing the agents' and the environment's models is illustrated in Fig. 2.12. Having modeled as ϵ_0-NFAs the capabilities, the constraints and the models of all agents, we proceed to construct the environment model, expressed as ϵ_0-NFA, which is the output of the modeling procedure.

2.2.3.1 Environmental Capabilities

We model the environmental capabilities as the concatenation of the agents' capabilities ${}^{\epsilon_0}K_i$, $i \in \{1, \ldots, n\}$, formed as in eq. (2.1), utilizing the concatenation operation of compatible automata to express the available environment state transitions. For the capabilities ${}^{\epsilon_0}K_i$ and ${}^{\epsilon_0}K_j$, of agents i and j, $i \neq j$ with $i, j \in \{1, \ldots, n\}$, let ${}^{\epsilon_0}K_i \asymp {}^{\epsilon_0}K_j$. The environmental capabilities are modeled by the ${}^{\epsilon_0}\mathcal{K}$ as:

$$ {}^{\epsilon_0}\mathcal{K} \triangleq \coprod_{i=1}^{n}{}_{\asymp} {}^{\epsilon_0}K_i. \tag{2.4} $$

2.2.3.2 Environmental Constraints

We model the environmental constraints as the concatenation of agents' constraints ${}^{\epsilon_0}D_i$, formed as in eq. (2.2), utilizing the concatenation operation of compatible automata to express the non-available environment state transitions. For the constraints ${}^{\epsilon_0}D_i$ and ${}^{\epsilon_0}D_j$, $i \neq j$ with $i, j \in \{1, \ldots, n\}$, let ${}^{\epsilon_0}D_i \asymp {}^{\epsilon_0}D_j$. The environmental constraints are modeled by ${}^{\epsilon_0}\mathcal{D}$ as:

$$ {}^{\epsilon_0}\mathcal{D} \triangleq \coprod_{i=1}^{n}{}_{\asymp} {}^{\epsilon_0}D_i. \tag{2.5} $$

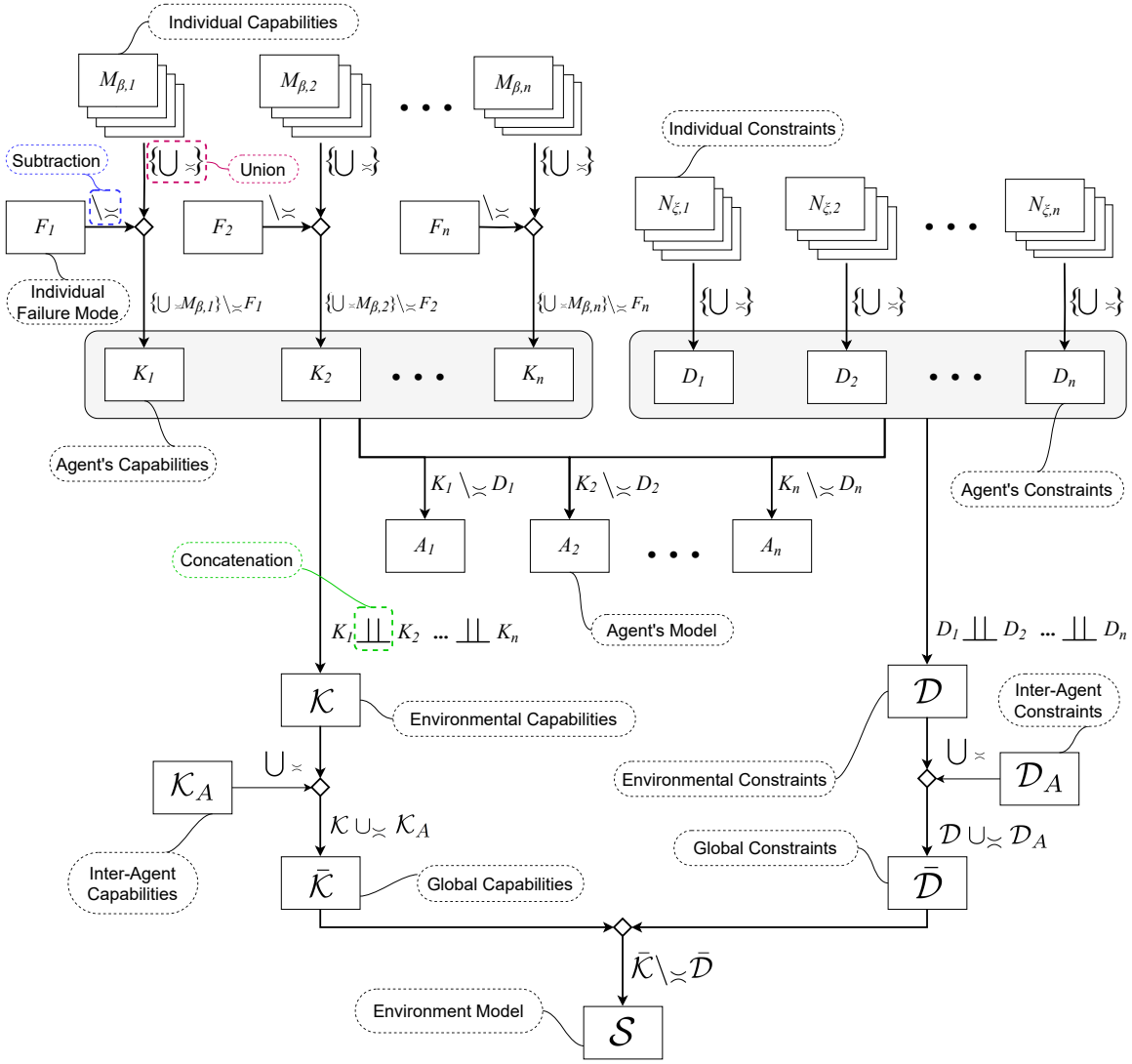


Figure 2.12: Flow diagram of the operations to produce the models of agents ${}^{\epsilon_0}A_i$ and the environment ${}^{\epsilon_0}\mathcal{S}$.

2.2.3.3 Inter-Agents Capabilities

Using the agents' capabilities formed as in eq. (2.1), the inter-agents capabilities ${}^{\epsilon_0}\mathcal{K}_A$ are modeled as follows: Let $\mathrm{co} \subseteq {}^{\epsilon_0}\mathcal{A}$ with $|\mathrm{co}| \leq n$ be a set of compatible agent models. Then,

$$ {}^{\epsilon_0}\mathcal{K}_A \triangleq \underset{i \in \mathrm{co}}{\amalg_{\asymp}} {}^{\epsilon_0}A_i $$

denotes the inter-agents capabilities between the members of co. Inter-Agents Failure Modes can be defined so as to restrict Inter-Agents Capabilities.

2.2.3.4 Inter-Agents Constraints

Using the agents' constraints formed as in eq. (2.2), the inter-agents constraints ${}^{\epsilon_0}\mathcal{D}_A$ are modeled as follows: Let co $\subseteq {}^{\epsilon_0}\mathcal{A}$ with $|\text{co}| \leq n$ be a set of compatible agent ϵ_0-NFAs. Then, the ϵ_0-NFA

$$ {}^{\epsilon_0}\mathcal{D}_A \triangleq \underset{i \in \text{co}}{\amalg_{\asymp}} {}^{\epsilon_0}A_i $$

denotes the inter-agents constraints between the members of co.

2.2.3.5 Global Capabilities

We model the global capabilities ${}^{\epsilon_0}\widetilde{\mathcal{K}}$ that are derived from the union of environmental capabilities ${}^{\epsilon_0}\mathcal{K}$ (formed as in eq. (2.4)) with inter-agent capabilities ${}^{\epsilon_0}\mathcal{K}_A$. For the environmental and inter-agent capabilities ${}^{\epsilon_0}\mathcal{K}$ and ${}^{\epsilon_0}\mathcal{K}_A$, let ${}^{\epsilon_0}\mathcal{K} \asymp {}^{\epsilon_0}\mathcal{K}_A$. The global capabilities are defined as:

$$ {}^{\epsilon_0}\widetilde{\mathcal{K}} \triangleq {}^{\epsilon_0}\mathcal{K} \cup_{\asymp} {}^{\epsilon_0}\mathcal{K}_A. \tag{2.6} $$

2.2.3.6 Global Constraints

We model the global constraints ${}^{\epsilon_0}\widetilde{\mathcal{D}}$ that are derived from the union of environmental constraints ${}^{\epsilon_0}\mathcal{D}$ (formed as in eq. (2.5)) with inter-agent constraints ${}^{\epsilon_0}\mathcal{D}_A$. For the environmental and inter-agent constraints ${}^{\epsilon_0}\mathcal{D}$ and ${}^{\epsilon_0}\mathcal{D}_A$, let ${}^{\epsilon_0}\mathcal{D} \asymp {}^{\epsilon_0}\mathcal{D}_A$. The global constraints are defined as:

$$ {}^{\epsilon_0}\widetilde{\mathcal{D}} \triangleq {}^{\epsilon_0}\mathcal{D} \cup_{\asymp} {}^{\epsilon_0}\mathcal{D}_A. \tag{2.7} $$

Considering ${}^{\epsilon_0}\widetilde{\mathcal{K}} \asymp {}^{\epsilon_0}\widetilde{\mathcal{D}}$, the environment model is modeled by the ϵ_0-NFA ${}^{\epsilon_0}\mathcal{S}$ after subtracting the ϵ_0-NFA ${}^{\epsilon_0}\widetilde{\mathcal{D}}$ from the ϵ_0-NFA ${}^{\epsilon_0}\widetilde{\mathcal{K}}$:

$$ {}^{\epsilon_0}\mathcal{S} \triangleq {}^{\epsilon_0}\widetilde{\mathcal{K}} \backslash_{\asymp} {}^{\epsilon_0}\widetilde{\mathcal{D}} \tag{2.8} $$

The cardinality of $X_{\mathcal{S}}$ is $\theta = \prod_{i=1}^{n} |X_{A_i}|$.

2.2.4 Task Specification

Let b be an n-bit binary indicating the states of the agents that we are interested in specifying and φ an ordered set representing the concatenation of the goal states of those agents. We define the task specification γ as the tuple:

$$ \gamma \triangleq (\varphi, b). $$

Then, a state $x_d \in X_{\mathcal{S}}$ satisfies the task specification γ, if and only if:

$$ proj(x_d, b) := \varphi \tag{2.9} $$

Depending on the cardinality of φ (i.e. how many bits are active in b), single or multiple tasks (objectives) can be concurrently specified in γ.

2.3 Formulation as a Module Composition Problem

Let us now consider the environment model ${}^{\epsilon_0}\mathcal{S}$ and assume that we would like to use $x_{0,\mathcal{S}} \in X_{\mathcal{S}}$ as the initial state of our system. Thus, $\mathcal{S} = \Delta({}^{\epsilon_0}\mathcal{S}, x_{0,\mathcal{S}})$. Let $P_{in} \in X_{\mathcal{S}}$ and $P_{out} \in X_{\mathcal{S}}$ be the non-empty finite set of input and output ports respectively, where $P = P_{in} \cup P_{out}$ and $P_{in} \cap P_{out} \neq \emptyset$. The module T_j is defined as:

$$T_j \triangleq \{p_j, e_j, q_j\} \tag{2.10}$$

where $p_j \in P_{in}$, $e_j \in E_{\mathcal{S}}$, $q_j \in P_{out}$ and $c(T_j) \triangleq g_{E_{\mathcal{S}}}(e_j)$ is the cost of implementing module T_j. Let $p_j := x_{in} \in X_{\mathcal{S}}$ and $q_j := x_{out} \in X_{\mathcal{S}}$. Then, the module T_j is defined as $T_j := \{x_{in}, e_j, x_{out}\}$ and it is shown in Fig. 2.13.

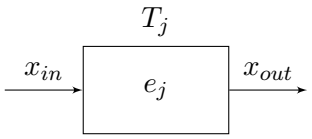


Figure 2.13: The module T_j.

Define the i^{th} task module as

$$T_{0,i} = \{x_{0,\mathcal{S}}, e_{0,i}, x_{d,i}\}$$

where

$$proj(x_{d,i}, b) := \varphi,\ x_{d,i} \in X_{\mathcal{S}}$$

and $e_{0,i}$ a virtual transition from the initial to the final state. Observe that there are up to $|\varphi|!$ potential solutions in the worst case, depending on the order that the goal state is reached by individual agents, so $i \in \{1, \ldots, |\varphi|!\}$. The task module $T_{0,i}$ is shown in Fig. 2.14.

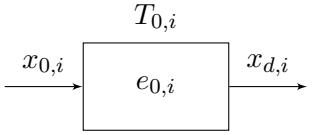


Figure 2.14: The task module $T_{0,i}$.

To tackle the problem defined in the Problem Statement, we proceed to formulate our problem as a Module Composition Problem (MCP) [85]. Since we are using single-port modules, we will have a special case of the MCP that is solvable in polynomial time.

We define $\mathcal{T}_i$ as the finite open module chain describing the task plan for a given task specification γ defined as:

$$\mathcal{T}_i = \{T_{1,i}, \ldots, T_{z,i}\} \tag{2.11}$$

where $z_i = |\mathcal{T}_i|$.

We define $\overset{\circ}{\mathcal{T}}_i$ as the finite closed module chain, containing $T_{0,i}^{-1}$ and task plan $\mathcal{T}_i$, describing the sequential environment states transitions during the execution (shown in Fig. 2.15) defined as:

$$\overset{\circ}{\mathcal{T}}_i = \{T_{0,i}^{-1}, \mathcal{T}_i\} \tag{2.12}$$

In the sequel we will drop the index i for notational brevity.

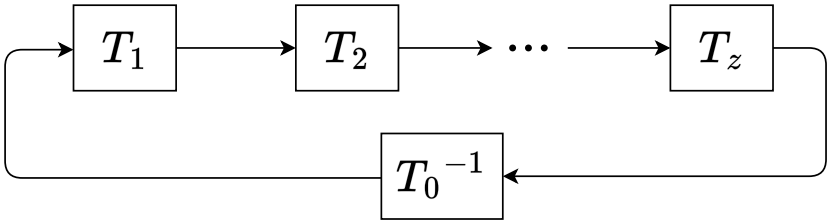


Figure 2.15: Closed module chain $\overset{\circ}{\mathcal{T}}$ with sequential single-port modules.

Let t_j denote the number of instances of T_j, $c_j = c(T_j)$ is the cost of implementing module T_j. Then, the discrete optimization problem can be posed as an integer programming problem as follows:

$$min \sum_{j \in \{1,\ldots,|E_{\mathcal{S}}|\}} c_j t_j \tag{2.13}$$

subject to:

$$\begin{aligned} & t_0 = 1, t_j \in \mathbb{Z}^+ \\ & \sigma_p = \sum_{q \in P_{out}} w_{p,q}, \forall p \in P_{in} \\ & \mu_q = \sum_{p \in P_{in}} w_{p,q}, \forall q \in P_{out} \end{aligned}$$

where $\mathbb{Z}^+$ denotes the non-negative integers, $w_{p,q}$ the number of connections between input port p and output port q, σ_p the number of modules with input port p utilized and μ_q the number of modules with output port q utilized. Since in the current work we have implemented only single port modules, the above integer programming problem can be reduced to the shortest directed path problem [85], that can be solved utilizing the Dijkstra's shortest dipath algorithm [5]. The optimal solution composes the $\mathcal{T}$ that minimizes the cost of state transitions in order to satisfy the task specification γ.

The limitation of using single port modules in the current work is that we are only limiting transitions to be performed by one agent at a time. Multi-port modules are currently being considered as a further research topic, to enable multiple agents to perform concurrent transitions. We have to also note here that the additional effort in casting the problem as a module composition one, enables us to seamlessly use the generated module chain as a model system for building supervisory controllers as in [80], as it is discussed in Part III Chapters.

2.4 Analysis

2.4.1 Pre-processing Analysis

The pre-processing step includes the construction of the agents models and the environment model (see Fig. 2.12) using the automata operations as described in sections 2.2.2 and 2.2.3. A polynomial amount of time on the number of agent states is required to construct the models of all agents and the environment model. More specifically, the model of agent ${}^{\epsilon_0}A_i$ is derived from the subtraction of agent's constraints from agent's capabilities. Both operations, the union and the subtraction, require a total of $O(|X_{A_i}|^2)$ time to be performed. Thus, the complexity of the agent model construction is $O(|X_{A_i}|^2)$. On the other hand, the concatenation operation requires $O(|X_{\mathcal{S}}|)$ time, where $|X_{\mathcal{S}}| = \prod_{i=1}^{n} |X_{A_i}|$, while both the union and the subtraction operation require $O(|X_{\mathcal{S}}|^2)$. Hence, the time complexity for the construction of the environment model is $O(|X_{\mathcal{S}}|^2)$ in total.

It is important to note here that the environment model ${}^{\epsilon_0}\mathcal{S}$ is only needed to be constructed once and can be used thereafter for all possible task specifications and initial states. ${}^{\epsilon_0}\mathcal{S}$ is converted to a weighted directed graph $\mathcal{H}_{\mathcal{S}} = (\mathcal{V}_{\mathcal{S}}, \mathcal{E}_{\mathcal{S}})$, where the set of nodes $\mathcal{V}_{\mathcal{S}}$ corresponds to the set of states $X_{\mathcal{S}}$, the set of edges $\mathcal{E}_{\mathcal{S}}$ is defined by $f_{\mathcal{S}}$ and the associated cost $g_{\mathcal{S}}$. $\mathcal{H}_{\mathcal{S}}$ is related to the directed graph produced by the module composition procedure, since by the definition of module in eq. (2.10), edges in $\mathcal{E}_{\mathcal{S}}$ serve as vertices of the module composition graph. In addition, due to the compatibility relation requirements, the vertices of each edge in $\mathcal{E}_{\mathcal{S}}$ are unique, hence the directed graph produced by the module composition procedure is the Line graph [91] $L(\mathcal{H}_{\mathcal{S}})$ of $\mathcal{H}_{\mathcal{S}}$. Any directed path, in $\mathcal{H}_{\mathcal{S}}$ with its associated cost, has an equivalent path of the same cost and the same event sequence in $L(\mathcal{H}_{\mathcal{S}})$. To see this consider that a path $v_1e_1v_2e_2 \dots v_ne_nv_{n+1}$ in $\mathcal{H}_{\mathcal{S}}$ will necessarily correspond to the path $e_1(q_1p_2)e_2 \dots (q_{n-1}p_n)e_n$ in $L(\mathcal{H}_{\mathcal{S}})$ where (q_ip_{i+1}) are the edges in $L(\mathcal{H}_{\mathcal{S}})$ corresponding to the connection between output port q_i and input port p_{i+1}. Hence solving a shortest directed path problem on $\mathcal{H}_{\mathcal{S}}$ is equivalent to solving the module composition problem (eq. 2.13).

Regarding complexity, the adjacency matrix representation of $\mathcal{H}_{\mathcal{S}}$ is a 2-dimensional array $X_{\mathcal{S}} \times X_{\mathcal{S}}$. Each element in the array stores the cost $g_{\mathcal{S}}$ related to the edge $f_{\mathcal{S}}(x \in X_{\mathcal{S}}, e \in E_{\mathcal{S}})$. The amount of space required to store the array is $O(|\mathcal{V}_{\mathcal{S}}|^2)$ in worst case.

2.4.2 Analysis of the Complete Solution

The problem solving phase encapsulates the synthesis of the optimal task plan and the integration of the individual agent failure modes. Dijkstra's algorithm is implemented over the weighted graph $\mathcal{H}_{\mathcal{S}}$ (Algorithm 1, lines 8-15) to find the optimal task plan $\mathcal{T}$ as given in Algorithm 1 in lines 21-23. In the case where a fault is detected for the transition from state x_i to x_j of $\mathcal{S}$ (e.g. by some fault identification system) that affects agent ν, we can disable all the affected transitions by finding the set of states X_i', X_j', where $proj(x_i, b_\nu) \equiv proj(x_i' \in X_i', b_\nu)$ and $proj(x_j, b_\nu) \equiv proj(x_j' \in X_j', b_\nu)$. Then, we eliminate the transitions from all $x_i' \in X_i'$ to all $x_j' \in X_j'$. New failure modes can be incorporated on-the-fly into ${}^{\epsilon_0}\mathcal{S}$ without the need to repeat the costly pre-processing step. The computational time required for this modification is $\theta_f = \prod_{i=1,\ i\neq\nu}^{n} |X_{A_i}|$ in the worst case.

The task planning problem of minimizing the length of the plan is NP-complete [92]. Let $\mathcal{P}$ be the

solution to this problem found after running Dijkstra's algorithm and let $\mathcal{P}_i$ denote it's i'th element, $i \in \{1, \ldots, |\mathcal{P}|\}$ (Algorithm 1, lines 16-20). In Algorithm 2, the optimal solution can then be found by running the algorithm for all states that satisfy the task specification (Algorithm 2, lines 3-16), that is $\theta' = \prod_{i=1,\ i\notin\sigma}^{n} |X_{A_i}|$ times in the worst case scenario, where σ denotes the set of agents that were used for the task specification γ. The running time for implementing Algorithm 1 with the "Complete" solution type (line 16-17), is $O(\theta'|\mathcal{E}_\mathcal{S}|\ log(|\mathcal{V}_\mathcal{S}|))$, considering the complexity of Dijkstra's algorithm [92]. This means that the "Complete" function depends on the number of states that satisfy the task specification as defined in eq. (2.9). Considering that $|\mathcal{E}_\mathcal{S}| < |X_\mathcal{S}||E_\mathcal{S}|$, that the event set is linear on the event sets of the individual agents, and that $\theta' < |X_\mathcal{S}|$, the worst case computational complexity of Algorithm 1 is bounded above by $O(|E_\mathcal{S}||X_\mathcal{S}|^2\ log(|X_\mathcal{S}|))$.

Algorithm 1 Create module chain $\mathcal{T}$.

Require: ϵ_0-NFA ${}^{\epsilon_0}\mathcal{S}$, initial state $x_{0,\mathcal{S}}$, task specification γ, solution type ST ("Complete" or "Heuristic")
Ensure: Module chain $\mathcal{T}$

1: Initialize $\mathcal{H}$ to be a zero matrix
2: Initialize E to be an empty cell array
3: Initialize $\mathcal{T}$ to an empty set
4: $\mathcal{S} \longleftarrow \Delta({}^{\epsilon_0}\mathcal{S}, x_{0,\mathcal{S}})$
5: **if** $proj(x_{0,\mathcal{S}}, b) := \varphi \wedge x_{0,\mathcal{S}} \in X_{m,\mathcal{S}}$ **then**
6: **return** $\mathcal{T}$
7: **else**
8: **for** $i \in \{1, \ldots, |X_\mathcal{S}|\}$ **do**
9: **for** $j \in \{1, \ldots, |X_\mathcal{S}|\}$ **do**
10: **if** $\exists e \in E_\mathcal{S} : f_\mathcal{S}(x_i, e) = x_j$ **then**
11: $\mathcal{H}[i][j] \longleftarrow g_\mathcal{S}(e)$
12: $E\{i\}\{j\} \longleftarrow e$
13: **if** ST = "Complete" **then**
14: [$\mathcal{P}$, cost] $\longleftarrow$ Complete($\mathcal{H}, \mathcal{S}$, $x_{0,\mathcal{S}}$, γ)
15: **else**
16: [$\mathcal{P}$, cost] $\longleftarrow$ Heuristic($\mathcal{H}, \mathcal{S}$, E, $x_{0,\mathcal{S}}$, γ)
17: **for** $i \in \{1, \ldots, |\mathcal{P}| - 1\}$ **do**
18: $\mathcal{T} \longleftarrow \mathcal{T} \cup \{\mathcal{P}_i,\ E\{\mathcal{P}_i\}\{\mathcal{P}_{i+1}\},\ \mathcal{P}_{i+1}\}$
19: **return** $\mathcal{T}$, cost

Algorithm 2 Complete Function.

1: **function** Complete($\mathcal{H}, \mathcal{S}, x_{0,\mathcal{S}}, \gamma$)
2: Initialize $cost_{min} \longleftarrow \infty$
3: **for** $i \in \{1, \ldots, |X_{\mathcal{S}}|\}$ **do**
4: **for** $j \in \{1, \ldots, |X_{\mathcal{S}}|\}$ **do**
5: **if** $proj(x_j, b) := \varphi \wedge x_j \in X_{m,\mathcal{S}}$ **then**
6: [$\mathcal{P}$, cost] $\longleftarrow$ Dijkstra($\mathcal{H}, x_{0,\mathcal{S}}, x_j$)
7: **if** $\mathcal{P} = \emptyset$ **then**
8: **return** Task infeasible.
9: **if** cost $< cost_{min}$ **then**
10: $cost_{min} \longleftarrow$ cost
11: $\mathcal{P}_{optimal} \longleftarrow \mathcal{P}$
12: **return** $\mathcal{P}_{optimal}, cost_{min}$

2.4.3 Analysis of a Heuristic Solution

In addition to the complete algorithm presented above, a heuristic approach is proposed in Algorithm 3 to reduce the computational time required to find a solution. By implementing Algorithm 1 with the proposed "Heuristic" solution type (line 19), reduces the complexity by only requiring a single execution of Dijkstra's algorithm, while sacrificing optimality and completeness. The computational time requirements of the proposed heuristic is $O(|\mathcal{E}_{\mathcal{S}}|\ log(|\mathcal{V}_{\mathcal{S}}|))$. Following similar arguments as in the complete solution, the worst case computational complexity of Algorithm 1 is bounded above by $O(|E_{\mathcal{S}}||X_{\mathcal{S}}|\ log(|X_{\mathcal{S}}|))$.

The intuition for utilizing the "Heuristic" solution type (Algorithm 3), is to avoid having the agents that are not participating in the task specification to end up in configurations different from their initial ones. Taking advantage of this, we solve the problem utilizing a goal state that satisfies the task specification where the agents not belonging to the projection have the same initial and final state. This does not imply that those agents are idle. Hence, we set the goal state x_d to be such that $proj(x_d, b) := \varphi$ and $proj(x_d, \overline{b}) = proj(x_{0,\mathcal{S}}, \overline{b})$ (Algorithm 3, line 3), where $\overline{b}$ denotes the bitwise negation. To discard unnecessary transitions and reduce the cost (recovering partial optimality), we track the solution to the minimum element k where $proj(\mathcal{P}_k, b) := \varphi$ and use the solution $\mathcal{P}^* = \{\mathcal{P}_1, \ldots, \mathcal{P}_k\}$ (Algorithm 3, line 9-15). While not possessing the completeness property, there are some cases where the proposed heuristic fully recovers optimality.

One such case is when b is only composed of ones, i.e. the full state vector of the goal state is specified. However, a classification of those cases and the conditions for feasibility of solutions is currently under consideration but is not part of the current work.

Algorithm 3 Heuristic Function.

function Heuristic($\mathcal{H}, \mathcal{S}$, E, $x_{0,\mathcal{S}}$, γ)
    Initialize x_d to an empty set
    Find $x_d \in X_{m,\mathcal{S}} : proj(x_d, b) := \varphi \wedge proj(x_d, \bar{b}) = proj(x_{0,\mathcal{S}}, \bar{b})$
    Initialize $c^* \longleftarrow 0$
    [$\mathcal{P}$, cost] $\longleftarrow$ Dijkstra($\mathcal{H}, x_{0,\mathcal{S}}, x_d$)
    if $\mathcal{P} = \emptyset$ **then**
        return No path to x_d.
    for $k \in \{2, \ldots, |\mathcal{P}|\}$ **do**
        $c^* \longleftarrow c^* + g_{\mathcal{S}}(E\{\mathcal{P}_{k-1}\}\{\mathcal{P}_k\})$
        if $proj(\mathcal{P}_k, b) := \varphi$ **then**
            $\mathcal{P}^* \longleftarrow \{\mathcal{P}_1, \ldots, \mathcal{P}_k\}$
            return $\mathcal{P}^*, c^*$

2.4.4 Completeness and Optimality

The principle of completeness asserts that the algorithm always returns a solution (if one exists), otherwise if there is no solution, the algorithm reports failure. We have the following results regarding the completeness of Algorithm 1 and Algorithm 2:

Proposition 1. *For a system constructed based on individual and inter-agent capabilities, constraints and failure modes of the multi-agent system as presented in Section 2.2, the resulting environment model of eq. (2.8) is complete, in the sense that it represents all and only those state transitions that are dictated by the automata capturing the individual and inter-agent capabilities, constraints and failure modes.*

Proof. To demonstrate the above claim we need to show that during the composition of ${}^{\epsilon_0}\mathcal{S}$, no valid transitions or states are being removed from the system and no new states or transitions are being introduced. In particular:

1. No states removed or added.

   This can be shown by observing that from Corollary 2 the ϵ_0-NFAs is closed under the operations of union, subtraction and concatenation.

   Union operation (Definition 7): This operates only on states defined in its arguments and the resulting states are the union of the argument's states that are part of agents' capabilities and constraints.

   Subtraction operation (Definition 8): This operation does not remove any states from the subtrahend argument.

   Concatenation operation (Definition 9): This operation creates the cross product state space of its argument. None of the operations introduces any state that does not already exist in their arguments.

2. No valid events are removed and no new transitions are introduced.

   Union and Concatenation operations: The resulting event set is the union of the operator argument events. No new events are introduced, that do not already exist in the arguments.

   Subtraction operation: The resulting event set includes only the events that are in the subtrahend and not in the subtractor's event set without introducing any new event. In particular failure mode

events are removed from individual agent's capabilities with subtraction operation without affecting any other events. The event set of global constraints is removed from the global capabilities without affecting events that are not included in the global capabilities event set.

Any event that is in the individual or inter-agent capabilities and is not in the failure modes, individual constraints or inter-agent constraints event sets will be included in the environment model. Any event that is in the failure modes event set (but not in the inter-agent capabilities) as well as any event in the individual or inter-agent constraint event set, will not appear in the environment model event set. Events that are not included in the individual and inter-agent capabilities will not appear in the environment model event set.

Hence all and only transitions that are valid will appear in the environment model's event set. □

With the completeness property in place we can now state the following result about the optimality properties of the proposed system:

Proposition 2. *For a system constructed based on individual and inter-agent capabilities, constraints and failure modes of the multi-agent system as presented in Section 2.2, and assuming that only one agent is allowed to operate at any time instant, the solution of Algorithm 1 with the "Complete" solution type of Algorithm 2, produces the optimal sequence of actions that brings the system from any initial state to a state that satisfies the task specification γ while minimizing the cost of the task plan $\mathcal{T}$.*

Proof. Since according to Proposition 1 the environment model is complete, then all (if any) optimal solutions are encoded in the transition system imposed by the transition function of the environment model. The module composition problem is an integer optimization problem and in our case maps the task planning problem to the shortest directed path - a graph search problem. The implemented Dijkstra's solution is both complete and is guaranteed to find an optimal solution if such a solution exists. □

2.4.5 Class of Addressable Problems

To assess the expressiveness of the proposed framework we have the following result:

Corollary 3. *If G is a DFA then it can be converted to the ϵ_0-NFA ${}^{\epsilon_0}G$ by introducing the state x_0 as the initial state of ${}^{\epsilon_0}G$ along with the associated ϵ transition to the starting state $x_{0,G}$ of G.*

Proof. This can be shown by observing that the six-tuple obtained by the operation is as the one in Definition 2. □

Based on Corollary 3, any DFA can be converted to ϵ_0-NFA. Furthermore, the Transition Cost Function g_G of Definition 4 that is associated to E_G renders the ϵ_0-NFAs to weighted automata [93]. Hence, the proposed framework inherits the expressiveness of weighted automata.

Remark 1. *The LTL-based and weighted automata-based frameworks are complementary [94] since even though they have an intersection, each one addresses problems that the other cannot address.*

2.5 Case Studies

The proposed methodology is a domain agnostic task planner methodology that can be applied in various applications upon determining the appropriate abstractions. The SuPErvisory Control Task plannER (*SPECTER*) software has been developed to realize the proposed framework and is used to analyze the case studies. To demonstrate the applicability of the proposed methodology, two case studies are presented related to manufacturing logistics workflows, implemented on a computer with AMD Ryzen 5 4500U 2.3 GHz and 16GB RAM. The case studies are motivated by experiments implemented in European Union's projects from companies belonging to the L4MS consortium. The time required to compute the solution is split between the time required for pre-processing (i.e. to construct ${}^{\epsilon_0}\mathcal{S}$) and the time required to compute either the complete or the heuristic solution. In both case studies the cost function captures the time for each transition and the task plans returned either by implementing the "Complete" or the "Heuristic" options were identical.

In first case study, we consider a non-trivial logistic problem of appropriate size that permits an effective illustration with task plan re-configuration when failure modes are detected. The cardinality of the environment state space in this case is $\theta_1 = \prod_{i=1}^{4} |X_{A_i}| = 560$ states.

The second case study pertains to a pattern of logistic workflow on manufacturing. This scenario is considered to face the challenges of productivity, efficiency and effectiveness of internal logistics management and agent automation in the workflow. In this case, the cardinality of the environment state space is $\theta_2 = \prod_{i=1}^{9} |X_{A_i}| = 1.875 \times 10^6$ states.

2.5.1 Case Study I

Considering that four agents are involved where ${}^{\epsilon_0}A_1$ models the robot R_1, ${}^{\epsilon_0}A_2$ models the robot R_2, ${}^{\epsilon_0}A_3$ models the worker W_1 and ${}^{\epsilon_0}A_4$ models the item I_1. Agents are on their initial state as shown in the plant of Fig. 2.16. The initial states of the agents are: the robot R_1 is at its docking station D, robot R_2 is at its docking station E, worker W_1 is at work-cell C, item I_1 is produced by machine at A.

Here, the objective is to have item I_1 at B in minimum time. Hence, the task planning problem is to find the optimal task plan $\mathcal{T}$ that takes the system from the initial state $x_{0,\mathcal{S}} = DECA$ to a state x_d, satisfying the task specification γ_1, while being subject to the provided capabilities and constraints. Formally, the task specification for this case study is defined as $\gamma_1 = (B, b_1)$, where $b_1 = 0001$ (i.e. single objective)

We define the capabilities and the constraints of ${}^{\epsilon_0}A_1, {}^{\epsilon_0}A_2, {}^{\epsilon_0}A_3, {}^{\epsilon_0}A_4$ as illustrated in Fig. 2.17 and Fig. 2.18. In Fig. 2.17, the states R_1 and R_2 of item's automaton capabilities ${}^{\epsilon_0}K_4$ (Fig. 2.17d) denote the payload of robot ${}^{\epsilon_0}A_1$ and robot ${}^{\epsilon_0}A_2$ (i.e. I_1 can be loaded to R_1 or R_2). The transitions of agent model ${}^{\epsilon_0}A_4$ are enabled through the inter-agent capabilities. Assuming that the robots should not visit the work-cell C, constraints of agents ${}^{\epsilon_0}A_1$ and ${}^{\epsilon_0}A_2$ are illustrated in Fig. 2.18.

To construct the model of agent ${}^{\epsilon_0}A_1$, agent's capabilities ${}^{\epsilon_0}D_1$ are subtracted from agent's constraints ${}^{\epsilon_0}K_1$. To construct the model of agent ${}^{\epsilon_0}A_2$, ${}^{\epsilon_0}D_2$ is subtracted from ${}^{\epsilon_0}K_2$. Considering that there are no constraints for agents ${}^{\epsilon_0}A_3$ and ${}^{\epsilon_0}A_4$, ${}^{\epsilon_0}D_3$ and ${}^{\epsilon_0}D_4$ are empty automata, hence ${}^{\epsilon_0}A_3 := {}^{\epsilon_0}K_3$ and ${}^{\epsilon_0}A_4 := {}^{\epsilon_0}K_4$. Using the concatenation operation on ${}^{\epsilon_0}K_1$, ${}^{\epsilon_0}K_2$, ${}^{\epsilon_0}K_3$ and ${}^{\epsilon_0}K_4$, the environmental capabilities automaton ${}^{\epsilon_0}\mathcal{K}$ is constructed. Using the concatenation operation ${}^{\epsilon_0}D_1$ and ${}^{\epsilon_0}D_2$, the environmental

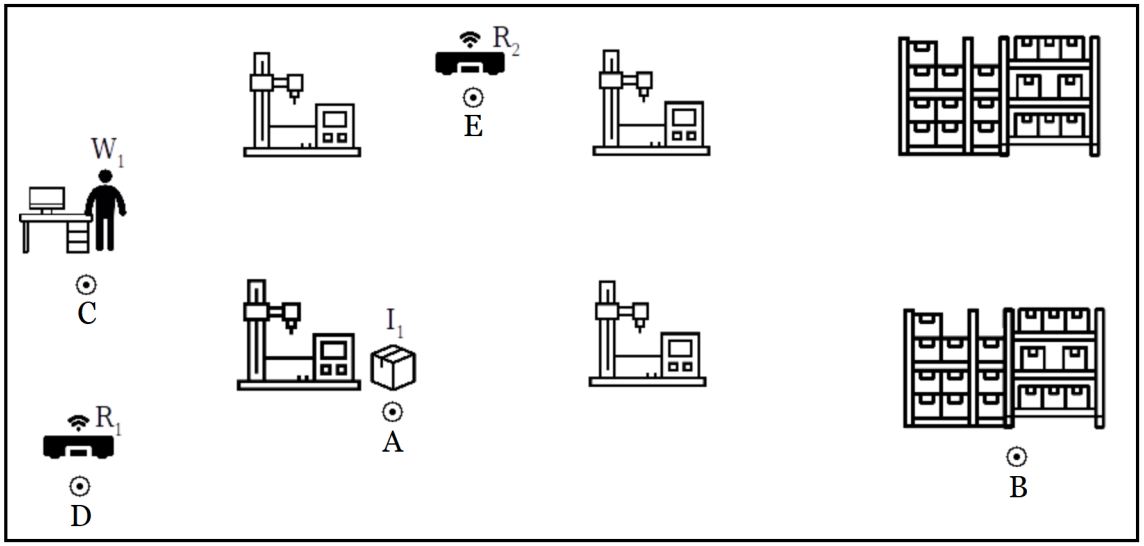


Figure 2.16: Case study I: Mock-up factory plant and agents.

constraints automaton ${}^{\epsilon_0}\mathcal{D}$ is constructed.

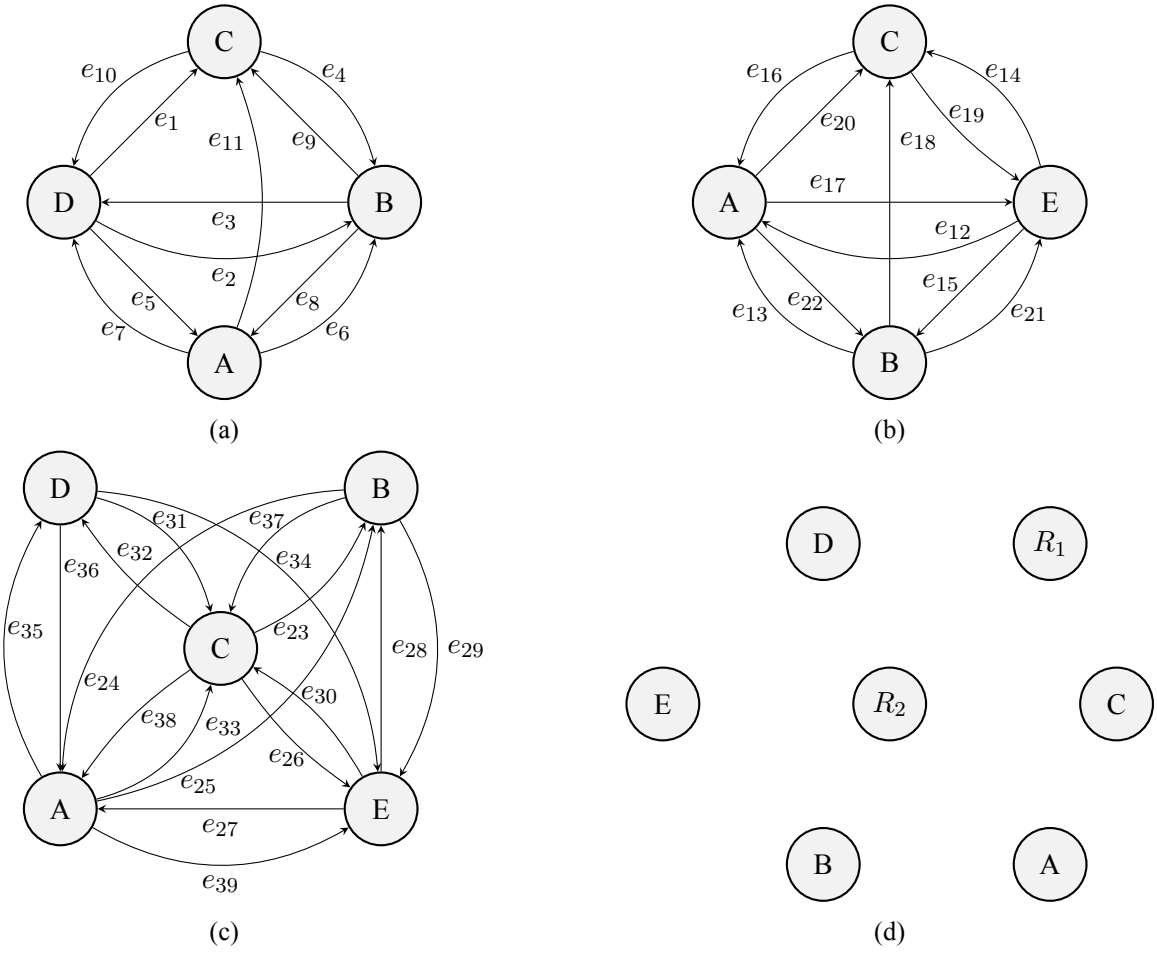


Figure 2.17: Case study I: State transition graphs of agents' capabilities. (2.17a) State transition graph of ${}^{\epsilon_0}K_1$, (2.17b) State transition graph of ${}^{\epsilon_0}K_2$, (2.17c) State transition graph of ${}^{\epsilon_0}K_3$, (2.17d) State transition graph of ${}^{\epsilon_0}K_4$.

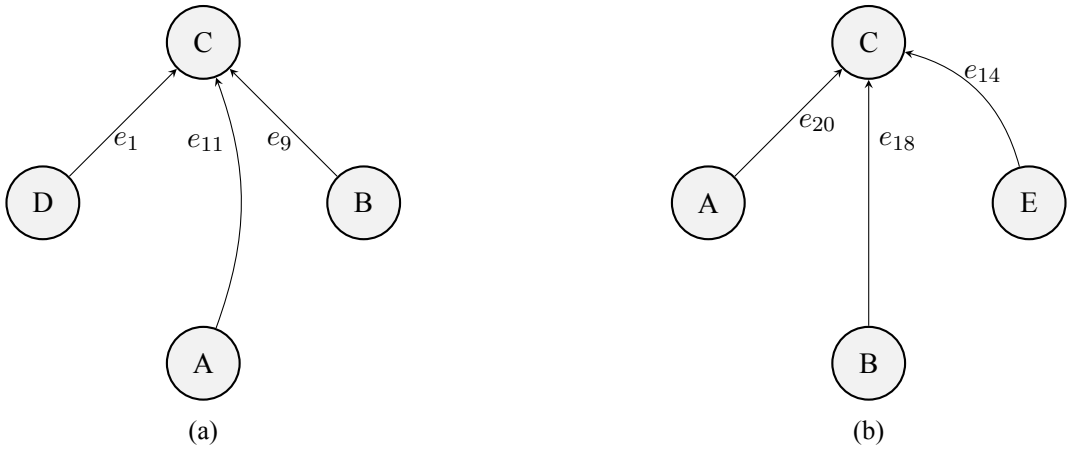


Figure 2.18: Case study I: State transition graphs of agents' constraints. (2.18a) State transition graph of ${}^{\epsilon_0}D_1$, (2.18b) State transition graph of ${}^{\epsilon_0}D_2$.

A sample of the environmental states transitions of ${}^{\epsilon_0}\mathcal{K}_A$ automaton are illustrated in Fig. 2.19. Inter-agent capabilities enable the loading and unloading actions of item to/from the robots, while defining the inter-agent capabilities between agents R_1, R_2, W_1, I_1. Taking into account the inter-agent capabilities to the environmental capabilities, the global capabilities automaton ${}^{\epsilon_0}\widetilde{\mathcal{K}}$ is constructed through the union operation on ${}^{\epsilon_0}\mathcal{K}_A$ and ${}^{\epsilon_0}\mathcal{K}$. Considering that there are no inter-agent constraints, ${}^{\epsilon_0}\mathcal{D}_A$ is an empty automaton, hence ${}^{\epsilon_0}\widetilde{\mathcal{D}} = {}^{\epsilon_0}\mathcal{D}$. Finally, the environment model ${}^{\epsilon_0}\mathcal{S}$ is constructed by subtracting ${}^{\epsilon_0}\widetilde{\mathcal{D}}$ from ${}^{\epsilon_0}\widetilde{\mathcal{K}}$, where $\theta_1 = |X_{\mathcal{S}}| = \prod_{i=1}^{4} |X_{A_i}| = 560$ states with $|X_{A_1}| = |X_{A_2}| = 4$ states, $|X_{A_3}| = 5$ states and $|X_{A_4}| = 7$ states. We form the task specification as $\gamma_1 = (B, b_1)$, where $b_1 = 0001$ (i.e. single objective). The task planning problem here is to find the optimal task plan $\mathcal{T}$ that takes the system from the initial state $x_{0,\mathcal{S}} = DECA$ to a state x_d, satisfying the task specification γ_1, in the minimum possible time, while being subject to the provided capabilities and constraints.

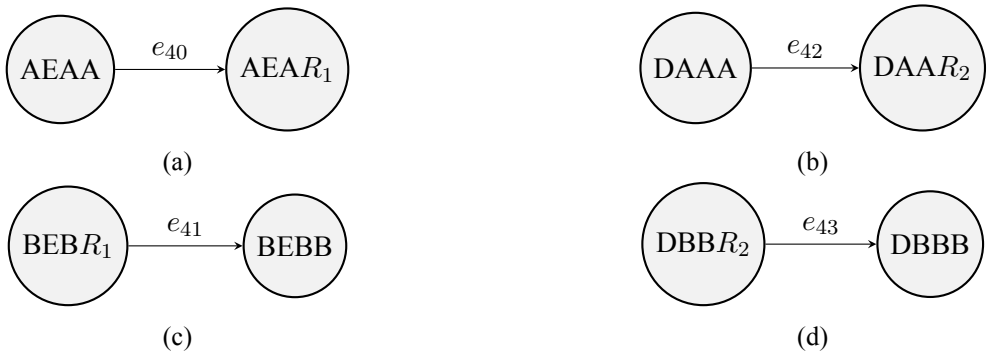


Figure 2.19: Case study I: State transition graphs of inter-agents capabilities. (2.19a) Loading item agent I_1 on robot agent R_1. (2.19b) Loading item agent I_1 on robot agent R_2. (2.19c) Unloading item agent I_1 from robot agent R_1 to Warehouse. (2.19c) Unloading item agent I_1 from robot agent R_2 to Warehouse.

The solution $\mathcal{T}$ computed by SPECTER with the "Heuristic" option results in a composition of a sequence of six modules and it's closed chain counterpart is presented in Fig. 2.20. In words, (T_1) R_2 navigates from E to the pick-up location A in 5 seconds, (T_2) W_1 goes from C to A to load the payload on R_2 in 7 seconds, (T_3) W_1 loads I_1 on R_2 in 3 seconds, (T_4), R_2 navigates from A to B in 12 seconds carrying the

payload, (T_5), W_1 goes to B in 17 seconds, and (T_6) W_1 unloads I_1 at B in 3 seconds. The time required for the agents involved to execute the task plan, so as to have item I_1 at B, is 47 seconds.

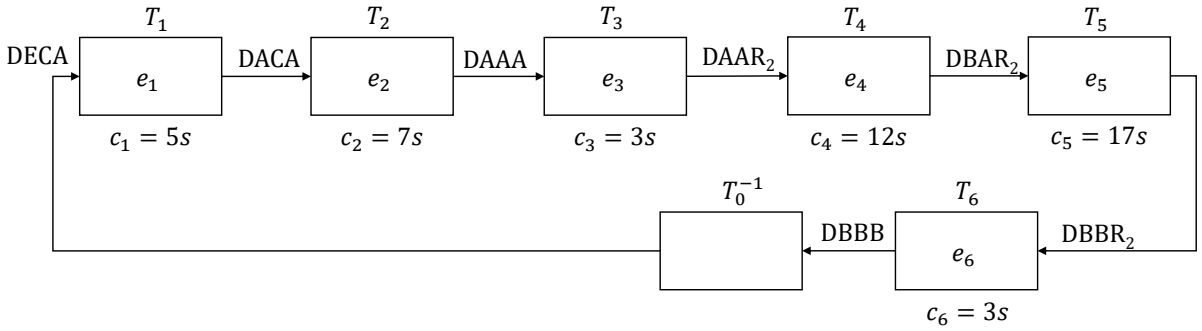


Figure 2.20: Case study I: The closed module chain $\overset{\circ}{\mathcal{T}}$ containing the task plan $\mathcal{T}$ for the task specification γ_1.

Regarding, the pre-processing runtime to construct the agents' and environment's models is 3.816×10^{-2} seconds. The problem solving runtime to find the solution $\mathcal{T}$ with the "Heuristic" option is 1.245×10^{-3} seconds whereas 5.952×10^{-2} seconds running the "Complete" option. SPECTER yields the same solution for the optimal $\mathcal{T}$ with the "Complete" option, for all 80 possible x_d's satisfying γ_1. Table 2.1 summarizes the runtime complexity of case study I.

Pre-processing	3.816×10^{-2} seconds
Complete solution	5.952×10^{-2} seconds for all 80 possible x_d's for which $proj(x_d, b_1) = B$
Heuristic solution	1.245×10^{-3} seconds

Table 2.1: Runtime complexity of case study I.

Now let us assume that a failure is detected on R_2 that renders R_2 unable to navigate from E to A (i.e. path of robot R_2 is blocked one way). The individual failure mode is modeled by the ${}^{\epsilon_0}F_2$ as shown in Fig. 2.21. The modified task plan $\mathcal{T}_f$ after incorporating failure mode ${}^{\epsilon_0}F_2$ is computed by SPECTER with the "Heuristic" option presented in Fig. 2.22. In words, (T_7) R_1 navigates from D to location A in 10 seconds, (T_8) W_1 goes from C to A in 7 seconds, (T_9) W_1 loads I_1 on R_1 in 3 seconds, (T_{10}) R_1 navigates from A to B in 15 seconds carrying the payload, (T_{11}) W_1 goes to B in 17 seconds, and (T_{12}) W_1 unloads I_1 at B in 3 seconds. The time required for the agents involved to execute the modified task plan is 55 seconds.

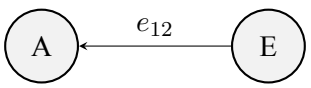


Figure 2.21: Case study I: State transition graph of failure mode of agent R_2.

The runtime to incorporate failure mode in the existing environment model is 7.89×10^{-4}. The problem solving runtime to find $\mathcal{T}_f$ with the "Heuristic" option is 1.131×10^{-3} seconds whereas 4.211×10^{-2} seconds are needed to find the optimal solution utilizing the "Complete" option. Table 2.2 summarizes the runtime complexity of case study I for task plan reconfiguration after incorporating the failure mode.

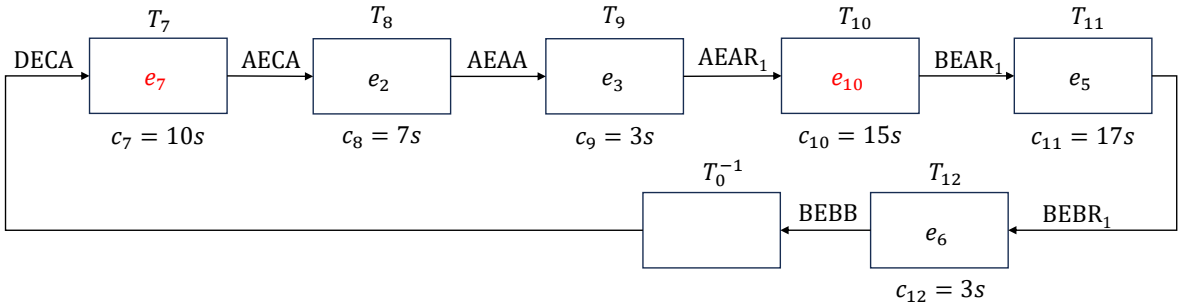


Figure 2.22: Case study I: The closed module chain $\mathring{\mathcal{T}}_f$ containing the modified task plan $\mathcal{T}_f$ for task specification γ_1, after incorporating failure mode ${}^{\epsilon_0}F_2$. Marked in red, are alternative events from $E_{\mathcal{S}}$, utilized to accommodate the failure.

Re-configuration pre-processing (to incorporate failure mode in the existing ${}^{\epsilon_0}\mathcal{S}$)	7.89×10^{-4} seconds
Complete solution	4.211×10^{-2} seconds for all 80 possible x_d's for which $proj(x_d, b_1) = B$
Heuristic solution	1.131×10^{-3} seconds

Table 2.2: Runtime complexity of case study I after incorporating failure mode in the existing environment model.

2.5.2 Case Study II

In this case study, we identified a pattern of two regular workflows concerning the manufacturing of "semi-finished products 1" and "semi-finished products 2" and "final products 1". The construction of "semi-finished products 1" require "raw materials 1" or "materials 2" whereas the "semi-finished products 2" require "raw material 3". The "semi-finished products 1" are produced by the "injection machine 1". The "semi-finished products 2" are created by "injection machine 2". Entities produced by injection machines are collected at buffer zones J or C. The "semi-finished products 1" are passed through the "conveyor 2" to the packing machine. The "final products 1" are created from "semi-finished product 1", which are passed through the "conveyor 2" to the packing machine. Packed entities are collected at buffer zone F and then are transported to Warehouse.

We consider that nine agents are involved in the workflow where ${}^{\epsilon_0}A_1, {}^{\epsilon_0}A_2, {}^{\epsilon_0}A_3$ model the "raw material 1", "raw material 2" and raw material 3, ${}^{\epsilon_0}A_4, {}^{\epsilon_0}A_5$ denote the "semi-finished product 1" and "semi-finished product 2", ${}^{\epsilon_0}A_6$ denotes the "final product 1", ${}^{\epsilon_0}A_7, {}^{\epsilon_0}A_8$ model "robot 1" and "robot 2" and ${}^{\epsilon_0}A_9$ models the human-worker. The initial states of the agents are shown in Fig. 2.23: the "raw material 1", "raw material 2", "raw material 3" are stored at the Warehouse, each robot is at its docking station and the human is at the Warehouse. The abstraction model of the workflow is illustrated in Fig. 2.24. In this case study we pose a multi-objective task specification with three concurrent objectives, as follows:

- Objective 1: Produce "final product 1" and store it at Warehouse
- Objective 2: Produce "semi-finished product 1"
- Objective 3: Produce "semi-finished product 2"

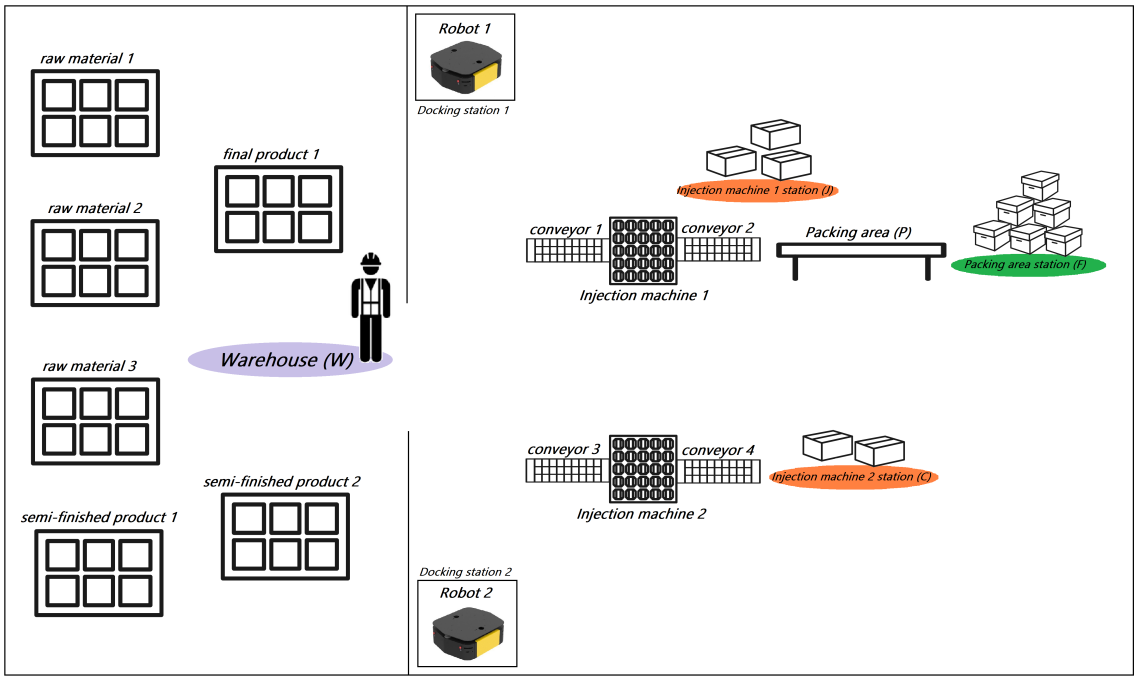


Figure 2.23: Case study II: Factory plant.

Formally, the multi-objective task specification for this case study with the three concurrent objectives is defined as $\gamma_2 = (JCW, b_2)$ with $b_2 = 000111000$.

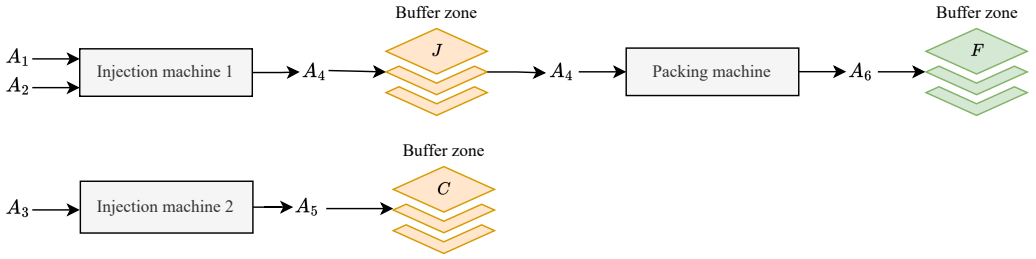


Figure 2.24: Workflow abstraction of case study II.

Models of ${}^{\epsilon 0}A_7$, ${}^{\epsilon 0}A_8$ and ${}^{\epsilon 0}A_9$ are constructed by combing the agents capabilities and constraints. The transitions of ${}^{\epsilon 0}A_1, {}^{\epsilon 0}A_2, {}^{\epsilon 0}A_3, {}^{\epsilon 0}A_4, {}^{\epsilon 0}A_5, {}^{\epsilon 0}A_6$ are enabled or disabled through the inter-agent capabilities and inter-agent constraints. Thus, ${}^{\epsilon 0}\mathcal{S}$ is constructed, where $\theta_2 = |X_{\mathcal{S}}| = \prod_{i=1}^{9} |X_{A_i}| = 1.875 \times 10^6$ states, with $|X_{A_1}| = |X_{A_3}| = |X_{A_4}| = |X_{A_5}| = |X_{A_6}| = |X_{A_7}| = |X_{A_8}| = 5$ states, $|X_{A_2}| = 6$ states and $|X_{A_9}| = 4$ states. The multi-objective task specification with the three concurrent objectives is defined as $\gamma_2 = (JCW, b_2)$ with $b_2 = 000111000$.

The solution $\mathcal{T}$ computed with the "Heuristic" option results in a composition of a sequence of twenty four modules. In words, (T_1) "raw materials 1", "raw materials 2", "raw materials 3" are stored at W, robots are at their docking stations, worker is at W, (T_2) "robot 1" goes to W, (T_3) worker loads the "raw material 3" to "robot 1", (T_4) worker loads "raw material 2" to "robot 1", (T_5) worker loads "raw material 1" to "robot 1", (T_6) worker goes to "injection machine 2", (T_7) "robot 1" goes to "injection

machine 2" while carrying raw materials, (T_8) worker unloads "raw material 3" from the "robot 1" and inserts "raw material 3" in "injection machine 2" at C, (T_9) "robot 1" goes to "injection machine 1" while carrying "raw material 1" and "raw material 2", (T_{10}) worker goes to "injection machine 1", (T_{11}) "injection machine 2" starts the production of "semi-finished product 2", (T_{12}) worker unloads "raw material 2" from "robot 1" and inserts the "raw material 2" in "injection machine 1" at J, (T_{13}) worker unloads "raw material 1" from "robot 1" and inserts "raw material 1" in "injection machine 1" at J, (T_{14}) "injection machine 1" starts the production of "semi-finished product 1", (T_{15}) "semi-finished product 1" is produced at "injection machine 1", (T_{16}) the "final product 1" is produced at F, (T_{17}) "semi-product 2" is produced by C, (T_{18}) worker goes to F, (T_{19}) "robot 2" goes to F, (T_{20}) worker loads "final product 1" on "robot 2", (T_{21}) "robot 2" goes to W, (T_{22}) "semi-finished product 1" is produced by "injection machine 1" at J, (T_{23}) worker goes to W, (T_{24}) worker unloads the "final product 1" from "robot 2" to W. The time required for the agents involved to execute the task plan, so as to satisfy the 3 concurrent objectives, is 42 hours.

The pre-processing runtime to construct the agents and environment models is 3.081×10^5 seconds. The problem solving runtime to find $\mathcal{T}$ utilizing Algorithm 3 is 1.657×10^2 seconds whereas with Algorithm 2 required a total of 1.72×10^6 seconds for all $15,000$ possible x_d's satisfying γ_2. Table 2.3 summarizes the runtime complexity of case study II. Note that in both case studies, the task plans provided by SPECTER, either by the "Complete" or "Heuristic" options, were identical.

Pre-processing	3.081×10^5 seconds
Complete solution	1.72×10^6 seconds for all 15,000 possible x_d's for which $proj(x_d, b_2) = JCW$
Heuristic solution	1.657×10^2 seconds

Table 2.3: Runtime complexity of case study II.

Part II

Discrete Abstractions

3 Discrete Abstractions for Manufacturing Logistics Optimization

The complexity of multi-agent systems analysis motivates the development of methods and tools that deal with the dimension and the exploitation of the complex system structure. The formal framework for modeling multi-agent systems utilizing automata theory is presented in Chapter 2. This framework supports description and analysis of multi-agent systems at varying levels of abstraction.

This chapter is an introduction to the construction of discrete abstraction models for utilization with the SPECTER task planner, the task planning framework presented in Chapter 2. A tutorial approach is used to demonstrate how abstraction models can be created. An application of the SPECTER task planner is presented so as to describe how the required discrete abstraction model of the manufacturing workflow is constructed. Several case studies are investigated, presenting the discrete abstraction models of a manufacturing workflow, and how different abstractions of the same workflow could reduce the model's state space.

Manufacturing logistics optimization addresses the improvement of the effectiveness of the overall performance of the factory and is based on reducing the appropriate costs through the manufacturing operations. Based on the abstraction model of a manufacturing workflow, the SPECTER task planner casts the manufacturing logistics optimization problems to the appropriate mathematical (combinatorial) optimization models over the specified objectives.

The rest of the Chapter is organized as follows: Section 3.1 presents the abstraction models of multi-agent systems. Section 3.2 presents the modeling of the workflow while Section 3.3 presents the environment and agents modeling using the ϵ_0-NFA formalism. Section 5.4 presents some case studies and Section 3.5 provides some discussions on the case studies results.

3.1 Abstraction model of multi-agent systems

The appropriate abstraction model needs to be determined so as to model the behavior of multi-agent systems. A generic abstraction is implemented where the system is abstracted by its capabilities, constraints and the agents involved in it, taking into account their interconnections. Automata also represent different levels of abstraction at which DESs are modeled and studied. An automaton is capable of representing a language according to well-defined rules of a multi-agent system. However, the choice of the appropriate abstraction level depends on the objectives of the analysis.

For a system constructed based on individual and inter-agent capabilities, constraints and failure modes of the multi-agent system as presented in Chapter 2, the abstraction model of the system needs to be constructed. In this chapter, the generic abstraction model is presented and a tutorial approach based on an application of the SPECTER task planner in an example from Food Service Industry is used to provide a detailed description of the abstraction models using automata.

The abstraction model consists of the following entities:

- *Environment*
  - *Areas of interest*:
    All possible areas that an agent could be.
  - *IDs of areas of interest*:
    Unique IDs correspond to the areas of interest.
- *Agents*:
  Entities that are acting on or interacting with the environment or trigger event(s).
- *Capabilities*:
  Individual capabilities of each agent as the available agent's transitions with associated costs.
- *Constraints*:
  Individual constraints of each agent as the non-available agent's transitions.
- *Inter-Agent Capabilities*:
  Capabilities that emerge when multiple agents are present with the associated costs.
- *Inter-Agent Constraints*:
  Constraints that emerge when multiple agents are present.
- *Current positions*:
  Starting position of agents.
- *Goal positions*:
  Goal position of agents.
- *Failure*:
  Failure occurred on an agent.

The template for encoding in SPECTER in Appendix I presents the abstraction model of a multi-agent system based on the agents involved in it, their individual capabilities, constraints, failure modes as well as the environmental capabilities and constraints.

To demonstrate the applicability of the abstraction model described above, an application of the SPECTER task planner is presented in the sequel.

3.2 Manufacturing workflow modeling

Considering the manufacturing workflow in Food Service Industry, the appropriate abstraction model is constructed. The objective of the Food Service Industry under consideration was to optimize the manufacturing workflow. Different scenarios were investigated depending on the available resources, robotic agents and temporal costs of the workflow processes in order to determine the optimal number of the agents required for the final product manufacturing and the sequence of actions that need to be performed by those agents. Through the SPECTER task planner implementation, the industry could identify a better agent allocation in the production line, while the optimum number of agents required to perform the necessary tasks is specified.

The diagram of the manufacturing workflow is presented in Fig. 3.1. In this work, a cellular layout arrangement entailing four different work-cells is considered. Each work-cell contains a certain number of machines, equipment and supplies. Each process of a work-cell requires a specific number of robotic agents for the processes to be performed. Based on the company's needs, the robotic agents involved in a work-cell are grouped in a team. Temporal costs associated to each work-cell refer to the production of one unit product per hour. Buffer zones are considered as the logistics facilities set up near the production work-cells that serve to temporarily store semi-finished or finished products. We considered four buffer zones, one for each work-cell. Products manufactured in a work-cell are stored at the buffer zone related to the work-cell. The maximum capacity of each buffer is two units.

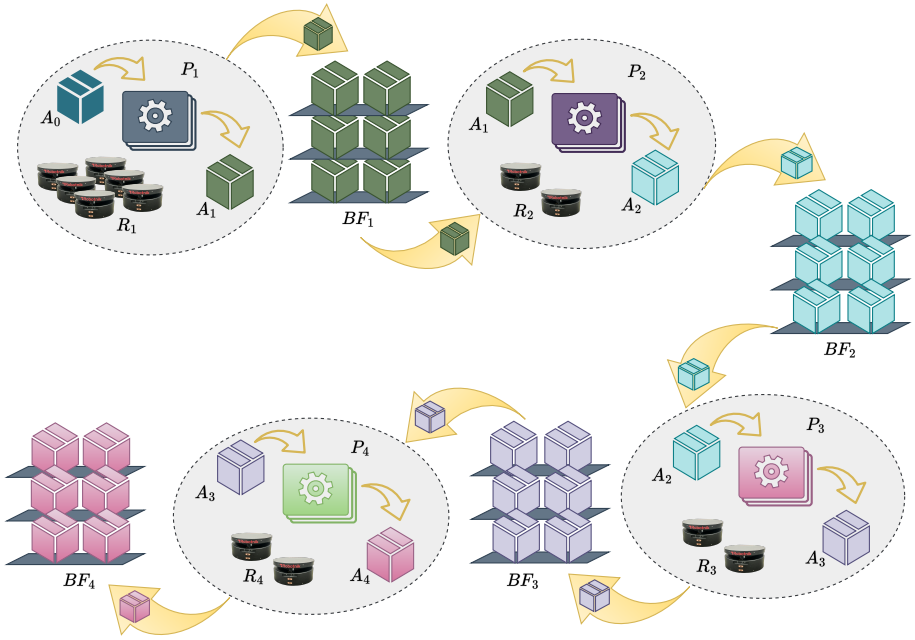


Figure 3.1: Diagram of the manufacturing workflow.

Let P_1, P_2, P_3 and P_4 represent the work-cells of the manufacturing workflow and BF_1, BF_2, BF_3 and BF_4 be the buffer zones related to each work-cell. We assume that the factory utilizes unprocessed material A_0 to construct the semi-finished products A_1, A_2, A_3 and manufacture the final product A_4. Moreover, BF_1 represents the buffer zone for semi-finished products A_1, BF_2 the buffer for A_2, BF_3 the buffer for A_3 and BF_4 the buffer for A_4. The company allots ten robots for this workflow. The minimum number of robots required for the production of one unit of A_4 is six.

The work-cells of the production workflow with its individual temporal costs, the number of robots involved in each work-cell, and the input and output products of each work-cell are presented in Table 3.1. The workflow process is considered as follows: In work-cell P_1, six robots are involved in order to produce one unit of semi-finished product A_1 after 7 hours using one unit of material A_0. The units of A_1 are stored at BF_1. In work-cell P_2, two robots are involved in order to produce one unit of semi-finished product A_2 after 4 hours using one unit of semi-finished product A_1 retrieved from BF_1. The units of A_2 are stored at BF_2. In work-cell P_3, two robots are involved in order to produce one unit of semi-finished

product A_3 after 5 hours using one unit of semi-finished product A_2 retrieved from BF_2. The units of A_3 are stored at BF_3. In work-cell P_4, two robots are involved in order to produce one unit of final product A_4 after 7 hours using one unit of semi-finished product A_3 retrieved from A_3. The units of A_4 are stored at BF_4.

Work-cell	P_1	P_2	P_3	P_4
Temporal cost	7	4	5	7
Number of robots	6	2	2	2
Input products	A_0	A_1	A_2	A_3
Output products	A_1	A_2	A_3	A_4

Table 3.1: Work-cells with costs and robots involved.

3.3 Abstraction model of manufacturing workflow

The abstraction model of the workflow is required to cast the problem in the discrete processes domain of the SPECTER framework. The agents considered for the workflow abstraction are: the robotic agents grouped in teams (i.e. each team is considered as an agent), buffer zones, materials, semi-finished and final products. The workflow abstraction is illustrated in Fig. 3.2. The nodes of the diagram represents the work-cells and buffer zones, whereas the arrows indicate the input and the output products of each work-cell and buffer zone.

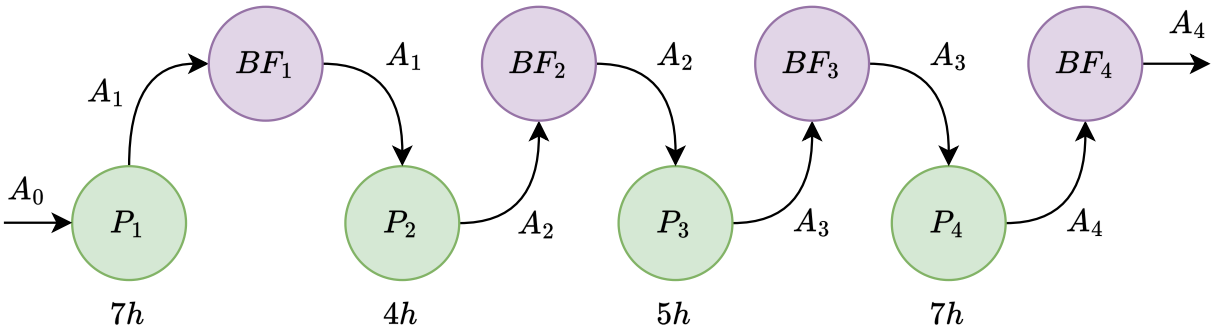


Figure 3.2: Workflow abstraction. Numbers below nodes indicate duration (hours).

Based on the workflow abstraction, the "Areas of interest" entity of the "Environment" is shown in Fig. 3.3.

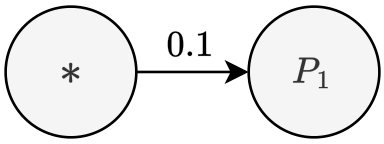
```
"Environment": {
  "Areas of interest": {
    "Name": [ "P1", "P2", "P3", "P4", "BF1", "BF2", "BF3", "BF4"],
    "Number": [ 1,   2,    3,    4,    5,     6,     7,     8  ]
  }
```

Figure 3.3: Encoding of "Areas of interest" entity.

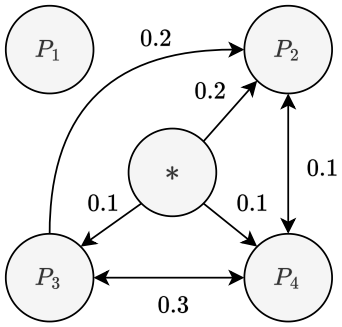
3.3.1 Agent modeling: Robots

The robotic agents could be abstracted in teams of either six or two agents. For the work-cell P_1, six robotic agents are required. Thus, a team of six robotic agents or three teams of two robotic agents are required for the work-cell P_1. Those two approaches determine two different abstractions, that will be exploited in the sequel. For the work-cells P_2, P_3 and P_4, one team of two robotic agents is required.

Let R_1 denote the team of six robots and R_2, R_3, R_4 denote the teams of two robots. The work-cells are considered as robots' states while the $*$ indicates that robots could be anywhere in the factory. The state set of the team R_1 is $X_{R_1} = \{*, P_1\}$, $X_{R_2} = \{*, P_1, P_2, P_3, P_4\}$ for R_2, $X_{R_3} = \{*, P_1, P_2, P_3, P_4\}$ for R_3 and $X_{R_4} = \{*, P_1, P_2, P_3, P_4\}$ for R_4. Thus, the state transition graphs of the capabilities of the robots teams are as depicted in Fig 3.4. The numbers on the edges represent the temporal costs for the transitions.



(a) State transition graph of capabilities of agent team R_1.



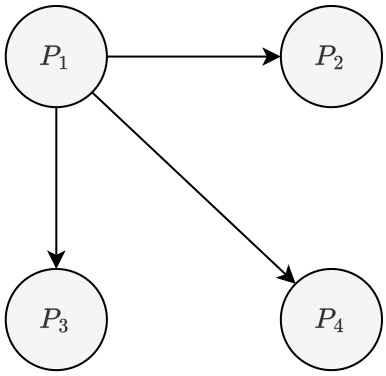
(b) State transition graph of capabilities of agent team R_2.

Figure 3.4: State transition graph of capabilities of robots teams.

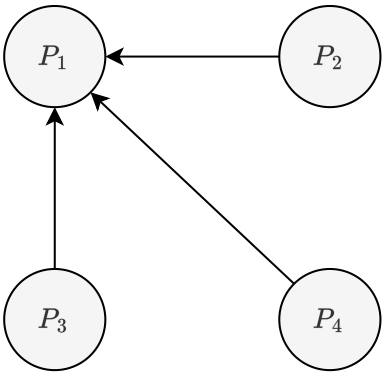
The capabilities of robots team R_1, shown in Fig. 3.4a, presents that the team R_1 could navigate at work-cell P_1 from anywhere in the factory. The capabilities of R_2, shown in Fig. 3.4b, presents that the team R_2 could navigate at work-cell P_2, P_3, P_4 from anywhere in the factory. The capabilities of R_3 and R_4 are considered the same as the capabilities of R_2.

The constraints of the robot teams are illustrated in Fig. 3.5. The constraints of robots team R_1, shown in Fig. 3.5a, presents that the team R_1 is not allowed to navigate and perform actions at P_2, P_3 and P_4, since the processes of work-cells P_2, P_3 and P_4 require exactly 2 robots to be performed. The constraints of robots team R_2, shown in Fig. 3.5b, presents that the team R_2 is not allowed to navigate and perform actions at P_1, since the processes of work-cell P_1 require exactly 6 robots in order to be performed. The constraints of R_3 and R_4 are considered the same as the constraints of R_2.

To construct the models of robots team R_1, the constraints ϵ_0-NFA of R_1 is subtracted from the capabilities ϵ_0-NFA of R_1. To construct the models of robots team R_2, the constraints ϵ_0-NFA of R_2 is subtracted from the capabilities ϵ_0-NFA of R_2. Repeating this procedure, the models of robot teams R_3 and R_4 are constructed.



(a) State transition graph of constraints of agent team R_1.



(b) State transition graph of constraints of agent team R_2.

Figure 3.5: State transition graph of constraints of robots teams.

3.3.2 Agent modeling: Materials, semi- and finished products

The state transition graphs of the material A_0, the semi-finished products A_1, A_2, A_3 and the final product A_4 are depicted in Fig. 3.6. The work-cells and buffer zones are considered as states. The transitions of A_0, A_1, A_2, A_3 and A_4 are enabled through the inter-agent capabilities. The state set of A_0 is $X_{A_0} = \{0, P_1\}$, $X_{A_1} = \{0, P_1, BF_1\}$ of A_1, $X_{A_2} = \{0, P_2, BF_2\}$ of A_2, $X_{A_3} = \{0, P_3, BF_3\}$ of A_3 and $X_{A_4} = \{0, P_4, BF_4\}$ of A_4. In the automata, the state ‘‘0” indicates that there is no available resources of the specific product.

An agent model is constructed by subtracting the agent’s constraints from the agent’s capabilities. We repeat the procedure for each agent to get the models of all agents. Utilizing the concatenation operation on the agent’s capabilities and constraints expressed as ϵ_0-NFAs, we construct the model of the environmental capabilities and environmental constraints expressed as ϵ_0-NFAs.

Let n be the total number of agents, then an environmental state consists of n elements such that a specific projection of an environment’s state indicates the state of each agent in any time instant (see Definition 10). We can now proceed with the modeling of inter-agent capabilities and inter-agent constraints.

(a) State transition graph of agent A_0.

(b) State transition graph of agent A_1.

(c) State transition graph of agent A_2.

(d) State transition graph of agent A_3.

(e) State transition graph of agent A_4.

Figure 3.6: State transition graphs of unprocessed material A_0 (Fig. 3.6a), semi-finished product A_1 (Fig. 3.6b), semi-finished product A_2 (Fig. 3.6c), semi-finished product A_3 (Fig. 3.6d), final product A_4 (Fig. 3.6e).

3.3.3 Agent modeling: Buffer zones

Fig. 3.7 models the state transition graph of a buffer zone in the production line. The transitions in the buffer zone model are enabled through the inter-agent capabilities. As shown in the code of Fig. 3.3, each area of interest has its own unique ID, such as the process P_1 has the ID #1 and the buffer zone BF_4 has the ID #8. There thirteen agents in total, where the material/products and buffer zones are considered also as agents. The materials/products are considered as agents, where its state space is all possible processes that the material could be utilized as input or the product could be produce. For example, the material A_0 could be utilized in process P_1, while the state 0 indicates that there are not available resources of material A_0 in the factory. Moreover, the buffer zones are considered as agents and their state space consists of three states: "0" indicates an empty buffer, "1" indicates that the buffer has one unit, "2" indicates that the buffer is holding two units, that is the capacity of this buffer. Thus, the "Agents" entity is shown in Fig. 3.8.

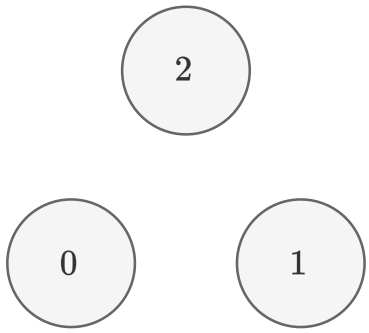


Figure 3.7: State transition graph of buffering agent BF_1.

```
"Agents": {
  "R_1": [ *, 1 ],
  "R_2": [ *, 1, 2, 3, 4 ],
  "R_3": [ *, 1, 2, 3, 4 ],
  "R_4": [ *, 1, 2, 3, 4 ],
  "A_0": [ 0, 1 ],
  "A_1": [ 0, 1, 5 ],
  "A_2": [ 0, 2, 6 ],
  "A_3": [ 0, 3, 7 ],
  "A_4": [ 0, 4, 8 ],
  "BF_1": [ 0, 1, 2 ],
  "BF_2": [ 0, 1, 2 ],
  "BF_3": [ 0, 1, 2 ],
  "BF_4": [ 0, 1, 2 ]
}
```

Figure 3.8: Encoding of "Agents" entity.

Using the encoding of areas of interest and agents, the individual capabilities and constraints, the inter-agent capabilities and constraints, current position of agents, goal position and failure are encoded in the sequel. The "Capabilities" entity, shown in Fig. 3.9, consists of the individual capabilities of the agents that are capable to navigate in the environment independently. Since the buffer zones and the products/materials can not be transported individually, then the agents' capabilities considered here are the capabilities of each robots team. The capabilities of buffer zones and materials/products are enabled through the inter-agent capabilities in the sequel.

```
"Capabilities": {
  "R_1": {
    "*": { "1":0.4 }
  },
  "R_2": {
    "*": { "2":0.2, "3":0.1, "4":0.1 },
    "2": { "4":0.1 },
    "3": { "2":0.2, "4":0.3 },
    "4": { "2":0.1, "3":0.3 }
  },
  "R_3": {
    "*": { "2":0.2, "3":0.1, "4":0.1 },
    "2": { "4":0.1 },
    "3": { "2":0.2, "4":0.3 },
    "4": { "2":0.1, "3":0.3 }
  },
  "R_4": {
    "*": { "2":0.2, "3":0.1, "4":0.1 },
    "2": { "4":0.1 },
    "3": { "2":0.2, "4":0.3 },
    "4": { "2":0.1, "3":0.3 }
  }
}
```

Figure 3.9: Encoding of "Capabilities" entity.

As shown in Fig. 2.17, the robot team R_1 could navigate from anywhere in the factory to the work-cell of process P_1 in 1 minute. Additionally, the robot team R_1 could navigate from the work-cell of process P_4 to the work-cell of process P_2 in 1 minute and to P_3 in 3 minutes.

The “Constraints” entity is shown in Fig. 3.10. As shown in Fig. 2.18, the robot team R_1 is not allowed to navigate from work-cells P_2, P_3, P_4 to work-cell of process P_1. Also, the robot teams R_2, R_3, R_4 could not navigate to work-cell of process P_1.

```
"Constraints": {
  "R_1": {
    "1": { "2", "3", "4" }
  },
  "R_2": {
    "2": { "1" },
    "3": { "1" },
    "4": { "1" }
  },
  "R_3": {
    "2": { "1" },
    "3": { "1" },
    "4": { "1" }
  },
  "R_4": {
    "2": { "1" },
    "3": { "1" },
    "4": { "1" }
  }
}
```

Figure 3.10: Encoding of “Constraints” entity.

3.3.4 Inter-agent capabilities

Inter-agent capabilities enable the processes procedures to construct the semi-finished and the final products, while defining the inter-agent capabilities between agents A_0, A_1, A_2, A_3, A_4, R_1, R_2. Assuming that the robots are grouped in four teams with the first being the team R_1 of six robots, the team R_2 being the second team of two robots, the team R_3 being the third team of two robots and R_4 being the fourth team of two robots. The capabilities and constraints of robots teams R_3 and R_4 are modeled as in Fig. 3.4b and Fig. 3.5b, respectively. A sample of the environmental states transitions are illustrated in Fig. 3.11. The agent's state indicated by $*$ means that the agent state is not considered in this inter-agent capability, in the sense of the agent state could take any value of the agent's state space.

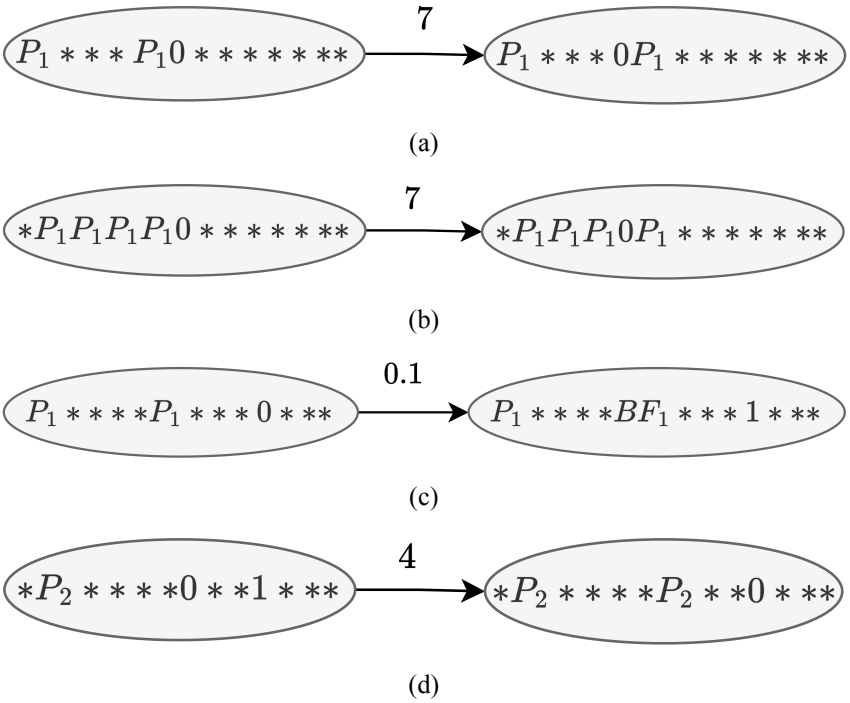


Figure 3.11: State transition graphs of inter-agents capabilities.

Fig. 3.11a models the production of semi-finished product A_1 utilizing team R_1 of six robots performing actions at work-cell P_1 using material A_0. The transition shows that A_1 is produced at P_1 after 7 hours. Fig. 3.11b models the production of semi-finished product A_1 utilizing material A_0 and engaging the robot teams R_2, R_3 and R_4, six robots in total grouped in three teams of two robots. This inter-agent capability enables the capability transition of R_2, R_2, R_3 to P_1 simultaneously. Fig. 3.11c models the loading of A_1 produced at P_1 at buffer zone BF_1 by team robots R_1 in 10 minutes. Fig. 3.11d models the production of semi-finished product A_2 in 4 hours using semi-finished product A_1 retrieved from buffer zone BF_1 utilizing the robots team R_2.

```
"Inter-Agent Capabilities": {
  "1": {
    "cost":[7],
    "A_0": [ 0, 1 ],
    "R_1": [ 1, 1 ]
  },

  "2": {
    "cost":[7],
    "R_2": [ 1, 1 ],
    "R_3": [ 1, 1 ],
    "R_4": [ 1, 1 ],
    "A_0": [ 1, 0 ],
    "A_1": [ 0, 1 ]
  },

  "3": {
    "cost":[0.1],
    "A_1": [ 1, 5 ],
    "BF_1": [ 0, 1 ],
    "R_1": [ 1, 1 ]

  },

  "4": {
    "cost":[4],
    "BF_1": [ 1, 0 ],
    "A_2": [ 0, 2 ],
    "R_2": [ 2, 2 ]
  },
  . . .
}
```

Figure 3.12: Encoding of “Inter-Agent Capabilities” entity.

Following the inter-agent capabilities shown in Fig. 3.11, a sample of the “Inter-Agent Capabilities” entity is shown in Fig. 3.12, where the inter-agent capability #1 models the Fig. 3.11a, the #2 models Fig. 3.11b, #3 models Fig. 3.11c and #4 models Fig. 3.11d.

3.3.5 Inter-agent constraints

Fig. 3.13 presents a sample of inter-agent constraints. The constraint of Fig. 3.13a states that the presence of all teams at work-cell P_1 at the same time is forbidden. Fig. 3.13b models the constraint of robots teams R_2, R_3 and R_4 to be present at P_2 at the same time.

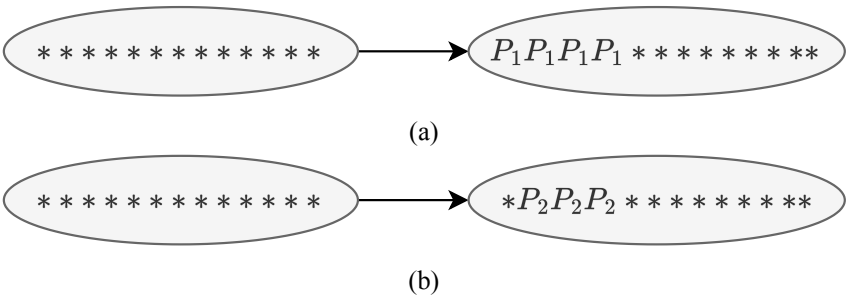


Figure 3.13: State transition graphs of inter-agent constraints.

Utilizing the union operation on the environmental capabilities and inter-agent capabilities, we construct the global capabilities. Utilizing the union operation on the environmental constraints and inter-agent constraints, we construct the global constraints. Subtracting global constraints from global capabilities through the subtraction operation, we construct the environment model ${}^{\epsilon_0}\mathcal{S}$ expressed as ϵ_0-NFA.

Having the environment model, we can state the objective (called the "task specification"). In our framework, that is defined as a projection of the environment's state space that indicates the desired state of specific agent(s) in the multi-agent system.

Following the inter-agent constraints shown in Fig. 3.13, a sample of the encoding of the "Inter-Agent Constraints" entity is shown in Fig. 3.14, where the inter-agent constraint #1 models the Fig. 3.13a and #2 models Fig. 3.13b.

```
"Inter-Agent Constraints": {
  "1": {
    "R_1": [ *, 1 ],
    "R_2": [ *, 1 ],
    "R_3": [ *, 1 ],
    "R_4": [ *, 1 ]
  },

  "2": {
    "R_2": [ *, 2 ],
    "R_3": [ *, 2 ],
    "R_4": [ *, 2 ]
  },
  . . .
}
```

Figure 3.14: Encoding of "Inter-Agent Constraints" entity.

3.3.6 Modeling of Starting state

The "Current positions" entity specifies the starting state of the agents interested in specifying at the time when the objective is set and the process for the optimal solution is started. In case that the starting state of an agent is not specified, then the optimal solution can then be found by running the SPECTER task planner for all possible starting states that satisfy the task specification. That is $ss' = \prod_{i=1,\, i \notin \sigma}^{n} |X_{A_i}|$ times in the worst case scenario, where σ denotes the set of agents that were used for the task specification γ, for which the objective is fulfilled.

The "Current position" entity that specifies all agents' starting positions is shown in Fig. 3.15.

```
"Current positions": {
    "R_1": [ * ],
    "R_2": [ * ],
    "R_3": [ * ],
    "R_4": [ * ],
    "A_0": [ 0 ],
    "A_1": [ 0 ],
    "A_2": [ 0 ],
    "A_3": [ 0 ],
    "A_4": [ 0 ],
    "BF_1": [ 0 ],
    "BF_2": [ 0 ],
    "BF_3": [ 0 ],
    "BF_4": [ 0 ]
}
```

Figure 3.15: Encoding of "Current positions" entity of all agents.

In this case, the current positions of the agents are: the robot teams could be anywhere in the factory floor, the materials/products are not available/produced yet, the buffer zones are empty.

Contrarily, the "Current position" entity that specifies the starting position of the agent interested in specifying is shown in Fig. 3.16.

```
"Current positions": {
    "R_1": [ * ],
    "R_2": [ * ],
    "R_3": [ * ],
    "R_4": [ * ],
    "A_0": [ 0 ]
}
```

Figure 3.16: Encoding of "Current position" entity of specific agents.

In this case, the current positions of the agents are: the robot teams could be anywhere in the factory floor, the materials are not available yet. The SPECTER task planner will provide optimal results utilizing all 24 possible starting states for which the objective is fulfilled.

3.3.7 Modeling of Objective

The task specification (see Section 2.2.4) defines the desired state of the agents. Having the task specification, the "Goal positions" entity specifies the goal state of the agents. In case that the objective defines the goal position of one agent, then the optimal solution can then be found by instructing the SPECTER task planner to provide solutions for all possible final states that satisfy the task specification. That is $\theta' = \prod_{i=1,\, i \notin \sigma}^{n} |X_{A_i}|$ times in the worst case scenario, where σ denotes the set of agents that were used for the task specification γ, for which the objective is fulfilled.

Considering that the objective is to produce two units of A_4 and store them at BF_4 utilizing robot team R_2. Then, the "Goal positions" entity is shown in Fig. 3.17.

```
"Current positions": {
    "R_2": [ 4 ],
    "BF_4": [ 2 ]
}
```

Figure 3.17: Encoding of "Goal position" entity.

In this case, the goal positions of the agents are: the robot team R_2 should be navigated at work-cell of process P_4 and the buffer zone BF_4 should contains two units of product A_4. The SPECTER task planner will run for all $218,000$ possible environment's final states for which the task specification is fulfilled.

3.4 Case studies

Two different case studies presenting the two different abstraction models were investigated. In each case study, different scenarios were implemented utilizing six, eight and ten robots in the production line. The case studies are motivated by experiments implemented in European Union's projects from companies belonging to the Better Factory consortium. Case studies were implemented on a computer with AMD Ryzen 5 4500U 2.3 GHz and 16GB RAM. In case studies A-I, A-II and A-III, the robots are grouped in teams of two robots, whereas in case studies B-I and B-II, the robots are grouped both in a team of six and teams of two robots. A time limit of one hour was set for the computation of the (sub-)optimal solution utilizing the heuristic solution option of Algorithm 3.

We assume that there are unlimited resources of unprocessed material A_0 at P_1. For the initial conditions, we assume that semi-finished products A_1, A_2 and A_3 and final product A_4 have not been produced yet. Buffer zones serve products transportation between work-cells. Robots are allowed to retrieve products from the buffer zones only.

Assuming that the buffer zones, the material and the products models are the same for all case studies, then the models of these agents are considered as follows. The state transition graphs of capabilities and constraints of each robots team are defined as in Fig. 3.4b and Fig. 3.5b, respectively. Each buffer zone is modeled as in Fig. 3.7, while the products A_0, A_1, A_2, A_3, A_4 are modeled as in Fig. 3.6. Using the agents' capabilities (Fig. 3.4a and Fig. 3.4b), agents' constraints (Fig. 3.5a and Fig. 3.5b), inter-agent capabilities (Fig. 3.11) and inter-agent constraints (Fig. 3.13), we construct the agents' and the environment model for each case study.

3.4.1 Case study A-I

For the first case study, we considered twelve agents; material A_0, semi-finished products A_1, A_2, A_3, final product A_4, buffer zones BF_1, BF_2, BF_3, BF_4 and six robots in total, grouped in three teams of two. The three teams together could perform actions in work-cell P_1, since six robots are required. However, each team is allowed to perform actions in work-cells P_2, P_3 and P_4, since two robots are required in each work-cell. Let R_2, R_3 and R_4 be the robot teams utilized. The objective is to produce two units of A_4 and locate it at $BM4$ utilizing six robots in the production line.

In the solution computed by SPECTER, the objective is fulfilled in twenty five steps. The capacity of buffer zones is presented in the Fig. 3.18. In the first step T_1, the teams R_2, R_3, R_4 goes at work-cell P_1. In step T_2, the process in P_1 is started utilizing material A_0. In T_3, A_1 is produced after 7 hours. In T_4, A_1 is stored at BF_1 (BF_1 capacity $= 1$ unit). In T_5, R_2 goes at P_2. In T_6, the process at P_2 is started utilizing one unit from BF_1 (BF_1 capac. $= 0$). In T_7, A_2 is produced at P_2 after 4 hours. In T_8, A_2 are stored at BF_2 (BF_2 capac. $= 1$ unit). In T_9, the teams R_2, R_3, R_4 goes at work-cell P_1. In T_{10}, the process in P_1 is started utilizing material A_0. In T_{11}, A_1 is produced after 7 hours. In T_{12}, A_1 is stored at BF_1 (BF_1 capacity $= 1$ unit). In T_{13}, R_2 goes at P_2, while R_4 goes at P_3. In T_{14}, the process at P_2 is started utilizing one unit from BF_1 (BF_1 capac. $= 0$), while the process at P_3 is started utilizing one unit from BF_2 (BF_2 capac. $= 0$). In T_{15}, A_2 is produced at P_2 after 4 hours, while A_3 is produced at P_3 after 5 hours. In T_{16}, A_2 are stored at BF_2 (BF_2 capac. $= 1$ unit). In T_{17}, A_3 are stored at BF_3 (BF_3 capac. $= 1$ unit), while R_4 goes at P_4. In T_{18}, procedure in P_4 is started utilizing A_3 from BF_3 (BF_3 capac.=0), while R_2 goes at P_3. In T_{19}, A_4 is produced after 7 hours, while A_3 is produced after 5 hours. In T_{20}, A_3 are stored at BF_3 (BF_3 capac. $= 1$ unit). In T_{21}, A_4 is stored at BF_4 (BF_4 capac. $= 1$ unit). In T_{22}, R_3 goes at P_4. In T_{23}, procedure in P_4 is started utilizing A_3 from BF_3 BF_3 (BF_3 capac.=0). In T_{24}, A_4 is produced after 7 hours. In T_{25}, A_4 is stored at BF_4 (BF_4 capac. $= 2$ units).

In this case, the cardinality of the environment state space is $1,640,250$ states. The time required to construct the environment and the agent models is 4.35×10^5 seconds. The runtime to calculate the solution is 1.004×10^3 seconds. In the case where there is a concurrent execution of the tasks in the task sequence which could be executed simultaneously by independent agents without affecting the fulfillment of the task specification, then the overall time for the task plan execution is reduced. Hence, the time required to fulfill the objective is 37 hours utilizing six robots for the whole process with concurrent task execution. The analysis of parallel execution of concurrent tasks in the task plan is part of the future work that is currently under investigation.

In case there is not a concurrent task execution, then the objective is fulfilled in thirty two steps while the time required to fulfill the objective is 46 hours. The robots are grouped in three teams of two. The first team R_2 of two robots will be utilized for 13 hours. The second team R_3 of two robots will be utilized for 12 hours. The third team R_4 of two robots will be utilized for 7 hours. Each team will be utilized for 14 hours more, since processes in P_1 requires six robot. So, that is 27 hours in total for the team R_2, 26 hours in total for the team R_3, 21 hours in total for the team R_4.

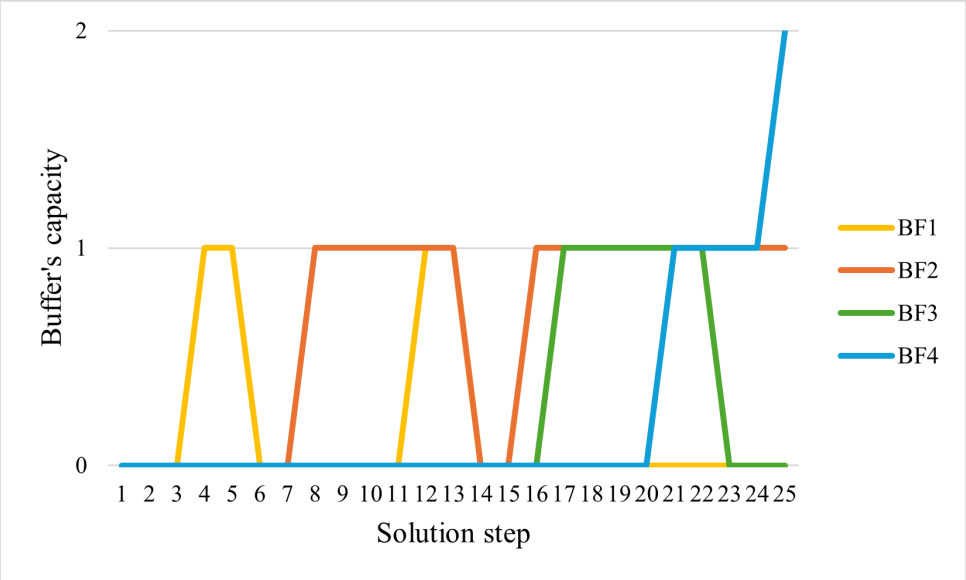


Figure 3.18: Buffer's capacity for the case study A-I.

3.4.2 Case study A-II

For the second case study A, we considered thirteen agents; material A_0, semi-finished products A_1, A_2, A_3, final product A_4, buffer zones BF_1, BF_2, BF_3, BF_4, eight robots in total grouped in four teams of two robots R_2, R_3 R_4, R_5. The three teams together could perform actions in work-cell P_1, since six robots are required. However, each team is allowed to perform actions in work-cells P_2, P_3 and P_4, since two robots are required in each work-cell. The capabilities and constraints of robots team R_5 are modeled as in Fig. 3.4b and Fig. 3.5b, respectively. The objective is to produce two units of A_4 and locate it at $BM4$ utilizing eight robots in the production line.

In the solution computed by SPECTER, the objective is fulfilled in twenty two steps. The three teams of two could handle the procedure in P_1, while all teams are able to handle the procedures in P_2, P_3 and P_4.

In this case, the cardinality of the environment state space is $8,201,250$ states. The time required to construct the environment and the agent models is $1,7 \times 10^6$ seconds. The runtime to calculate the solution is $3,6 \times 10^3$ seconds.

The time required to fulfill the objective is 25 hours utilizing 8 robots grouped in four teams of two robots for the whole process. The robots of teams R_2, R_3 and R_4 will be utilized for 14 hours to perform processes in P_1. The team R_3 will be utilized for 10 hours more to perform actions in P_3. The robots of team R_5 will be utilized for 22 hours.

3.4.3 Case study A-III

For the third case study A, we considered fourteen agents; material A_0, semi-finished products A_1, A_2, A_3, final product A_4, buffer zones BF_1, BF_2, BF_3, BF_4, ten robots in total grouped in five teams of two robots, R_1, R_2, R_3, R_4 and R_5. The objective is to produce two units of A_4 and locate it at $BM4$ utilizing ten robots in the production line. The objective is to produce two units of A_4 and locate it at $BM4$ utilizing ten robots in the production line. In this case, the cardinality of the environment state space is $41,006,250$ states and the runtime exceeded the computational time limit.

3.4.4 Case Study B-I

In the case study B-I, we considered eleven agents; material A_0, semi-finished products A_1, A_2, A_3, final product A_4, buffer zones BF_1, BF_2, BF_3, BF_4, eight robots in total grouped in the team R_1 of six robots and team R_2 of two robots. The team R_1 could perform action only in work-cell P_1, whereas R_2 is allowed to perform actions in work-cells P_2, P_3 and P_4. The objective is to produce two units of A_4 and locate it at $BM4$ utilizing eight robots in the production line.

In the solution computed by SPECTER, the objective is fulfilled in twenty five steps. The team R_1 is handling the procedure in P_1, while the team R_2 is handling the procedures in P_2, P_3 and P_4. The time required to fulfill the objective is 39 hours. The robots of R_1 will be utilized for 14 hours, while the robots of R_2 will be utilized for 32 hours.

The cardinality of the environment state space is $131,220$ states. The time required to construct the environment and the agent models is 6.27×10^3 seconds. The runtime to calculate the solution is 72 seconds.

3.4.5 Case study B-II

In the case study B-II, we considered thirteen agents; material A_0, semi-finished products A_1, A_2, A_3, final product A_4, buffer zones BF_1, BF_2, BF_3, BF_4, ten robots in total grouped in one team R_1 of six robots, two teams R_2 and R_3 of two robots. The objective is to produce two units of A_4 and locate it at $BM4$ utilizing six robots in the production line. The objective is to produce two units of A_4 and locate it at $BM4$ utilizing ten robots in the production line.

In the solution computed by SPECTER, the objective is fulfilled in twenty two steps. The team R_1 is handling the procedure in P_1, while the teams R_2 and R_3 are handling the procedures in P_2, P_3 and P_4. The time required to fulfill the objective is 25 hours. The team R_1 will be utilized for 14 hours, the team R_2 will be utilized for 22 hours and the team R_3 will be utilized for 10 hours.

The cardinality of the environment state space is $656,100$ states. The time required to construct the environment and the agent models is $2,4 \times 10^3$ seconds. The runtime to calculate the solution is 371 seconds.

3.5 Discussions

The outline of the scenarios parameters are summarized in the Table 3.2. The chart of Fig. 3.20 illustrates the production time for each case study as described in the section 5.4.

Case Study	No. of agents	No. of teams of 6 robots	No. of teams of 2 robots	State Space (states)	Time Costs (hours)
A-I	12	0	3	1,640,250	37
A-II	13	0	4	8,201,250	25
A-III	14	0	5	41,006,250	Runtime limit exceeded
B-I	11	1	1	131,220	39
B-II	13	1	2	656,100	25

Table 3.2: Case studies parameters.

As expected, a decrease in the environment state space is observed when the abstraction with the team of 6 robots is utilized. Analyzing the results of the case studies, there are tasks in the task sequences that are executed independently without affecting the fulfillment of the task specification. These tasks seem to have a synchronization point in the inter-agent tasks so that they can be performed in parallel. This is an interesting topic for further research and methodology expansion. In Fig. 3.19, the time required to produce two units of final product engaging six robotic agents requires 46 hours with sequential task execution whereas 37 hours are needed with concurrent task execution.

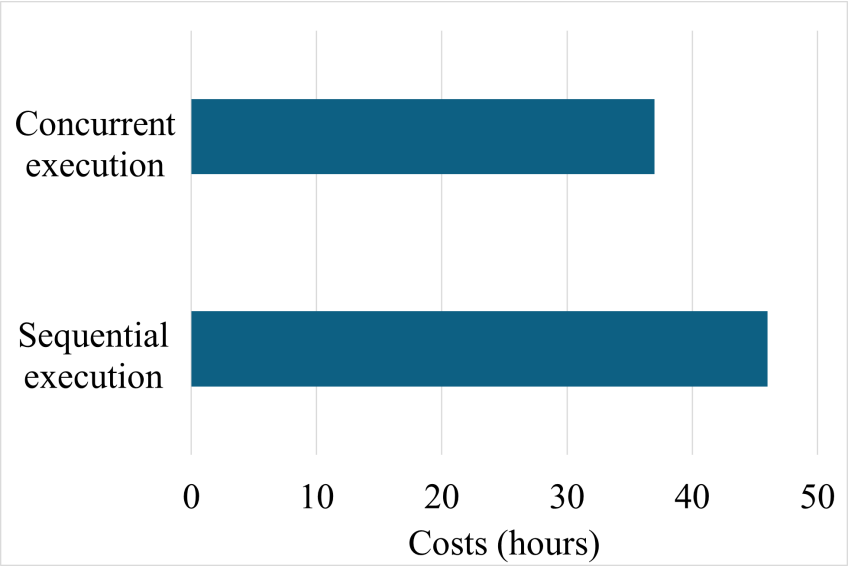


Figure 3.19: Costs for the case study A-I engaging 6 robotic agents.

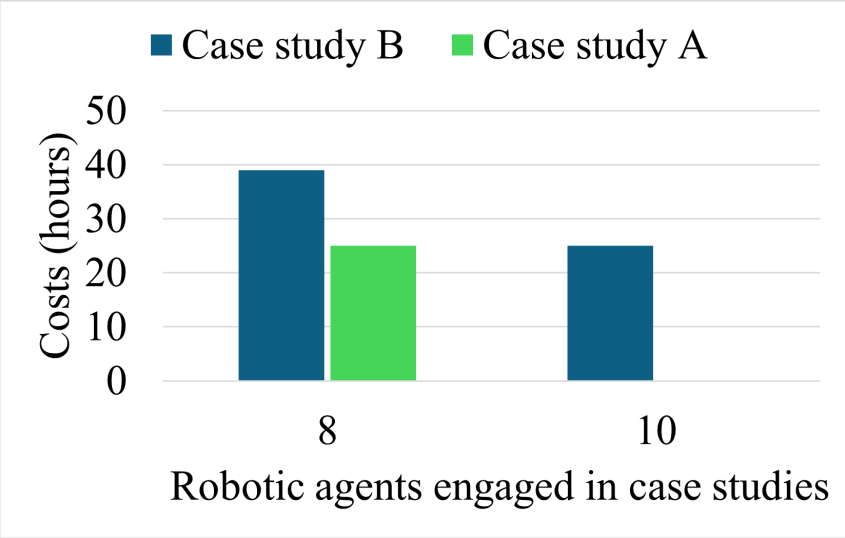


Figure 3.20: Costs (hours) for the case study A-II, case study B-I and case study B-II depending on the number of robotic agents utilized in each case study.

In Fig. 3.20, results of case studies A and B are depicted. Two different workflow abstractions are constructed engaging eight and ten robotic agents in the production line. A reduction in production time is observed when the robotic agents are grouped in teams of two agents while an increase in the state space is registered. Interestingly a decrease in the production time was observed with eight robots in the production line but no changes were registered when ten robotic agents engaged. Also, the different abstractions of the workflow do not affect the time required for the production of the final product. This concludes that the optimal (with respect to our case studies) number of robots needed for the production of two units of the final product are eight robots, requiring 25 hours.

4 Manufacturing Logistics Optimization using the SPECTER Task Planner: A Shoe Manufacturing Logistics Case Study

This chapter presents an application of SPECTER task planner for shoe manufacturing logistics. The case study is motivated by experiments implemented in European Union's projects from companies belonging to the Better Factory consortium, TAPI NERO® shoe manufacturing. This work focuses on the optimization of the final assembly workflow and its individual steps of a shoe manufacturing industry in order to optimize the intra-factory logistics. The challenge is that a single production line consists of many intermediate steps and multiple types of production processes depending on the customer order.

Buffer zones are the industrial facilities for storage and picking areas, so that materials/products can be retrieved quickly and efficiently while ensuring the safety of the people working in that area. In manufacturing logistics, buffer zones are widely used as a method to ensure the balance in the production processes, if there are disruptions in the supply chain. Moreover, buffer zones enable the parallel execution of multiple processes while collecting since they provide the capability of collecting materials/products from many processes. Furthermore, the optimal allocations of buffers can significantly improve the manufacturing process by managing the products' variability. Last but not least, the buffer capacity has an important role in the manufacturing process especially in the case of processes disruptions. The higher the buffer capacity is, the longer it will take for the production to be interrupted. This is the case for the company where buffer zones are needed to be utilized in several stages of the production line to store and retrieve materials and products fast and easily.

Utilizing the SPECTER task planning framework as described in Chapter 2, we develop an abstract model of the workflow of a shoe manufacturing company taking into account:

- the workflow stages;
- the time costs (i.e. machine operation, production time, worker transitions between work-cells);
- the agents involved in the production line (such as machines, humans, buffer zones, materials, intermediate products etc.).

Based on the derived abstraction, different scenarios are studied, to determine the resources and the time required for shoe production depending on the available human workers in the factory as well as the sequence of actions that need to be performed (i.e. the machines/robots operation sequence, the materials utilization, worker actions). Optimal solutions are provided for utilizing one, two or three workers in the production line. The chapter validates the modeling power of the SPECTER framework, while demonstrating its potential applications in providing solutions for the industry. The presented results depend on the assumptions, the accuracy of the data and on the level of abstraction and should be considered as a preliminary result that demonstrates the capabilities of our modeling framework.

The rest of the Chapter is organized as follows: Section 4.1 presents the description of the workflow, and the workflow abstract model, while Section 4.2 presents the results of three different scenarios. Section 4.3 concludes the chapter discussing the case study results.

4.1 Analysis and Modeling

4.1.1 Description of the Final Assembly Workflow

Final assembly is the workflow where upper leather is shaped, lasted and assembled with the sole to get the final product i.e. the shoes. The final assembly is performed before the packing procedure.

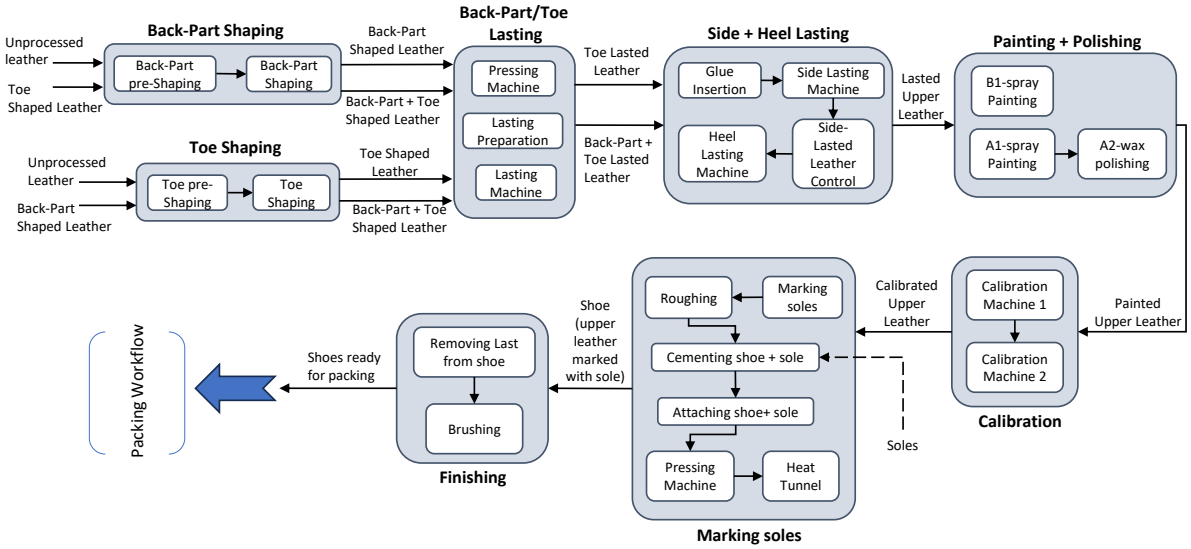


Figure 4.1: Diagram of the final assembly workflow of TAPI NERO® shoe manufacturing.

The main techniques of final assembly used in the shoe manufacturing industry, presented in Fig. 4.1 are:

- **Toe Shaping:** Unprocessed leather or back-part shaped leather is doubled shaped for preserving the shape and the original appearance of the shoe.
- **Back-Part Shaping:** Unprocessed leather or toe shaped leather is doubled shaped for preserving the shape and the original appearance of the shoe.
- **Lasting:** The unprocessed leather or back-part shape leather or back-part and toe shaped leather can be processed. In this stage, the upper leather is attached to the bottom of the shoe.
- **Side and Heel Lasting:** Sets the final shape of the shoe and holds it in place so the out-sole can be permanently attached.
- **Painting and Polishing**
- **Calibration:** Calibrating the shoe last.
- **Marking soles:** Marking the sole in the upper lasting leather, then roughing and cementing the shoe with the sole. When the shoe with the sole are attached, the shoe is pressed and then passed through the heat tunnel.
- **Finishing:** Removing last from shoe's inside, then brushing. Finally, the shoe is produced and continues to packing workflow.

4.1.2 Abstract Model

In order to model the workflow shown in Fig. 4.1 an abstraction is required to cast the problem in the discrete processes domain of the SPECTER framework. The abstraction of workflow is depicted in Fig. 4.2. The nodes of the diagram represents the processing steps and buffer zones, whereas the arrows indicate the input and the output products of each process and buffer zone.

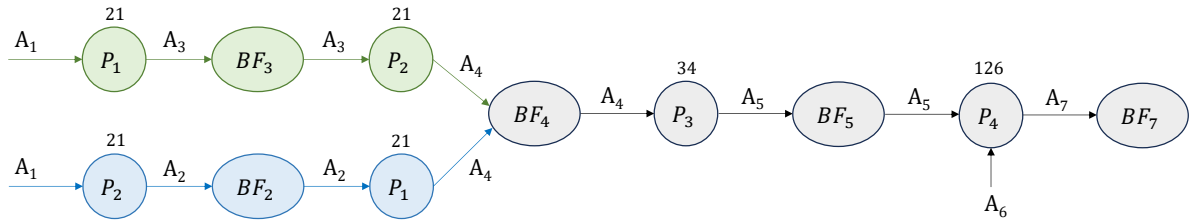


Figure 4.2: Workflow abstraction. Numbers above circles indicate duration (seconds).

Workers and items are modeled as agents expressed as ϵ_0-NFAs (see Chapter 2). More specifically, the unprocessed leather is modeled as A_1; the toe-shaping leather is modeled as A_2; the back-part shaping leather is modeled as A_3; the back-part and toe-shaping leather is modeled as A_4; the lasted leather is modeled as A_5; the soles are modeled as A_6; and the shoes are modeled as A_7. Additionally, three workers are considered working in the factory and modeled as A_8, A_9, A_{10} respectively.

Step P_1 represents the back-part shaping, P_2 the toe shaping, P_3 the back-part and toe lasting processes. The processes of side and heel lasting, painting and polishing, calibration, marking with soles and finishing are grouped in step P_4 since no sequence changes are allowed. Moreover, the buffer zones are considered as agents and their state space consists of four states: "0" indicates an empty buffer, "1" indicates that the buffer has one unit, "2" indicates that the buffer is holding two units, that is the capacity of this buffer and "3" indicates that the buffer is holding three units. More specifically, BF_2 represents the buffer zone for items modeled as A_2, BF_3 the buffer for A_3, BF_4 the buffer for A_4, BF_5 the buffer for A_5 and BF_7 the buffer for A_7. For the item agents A_1, A_2, A_3, A_4, A_5 and A_7, the processing steps and the buffer zones are considered as agents' locations. The BF_5 buffer zone is a conveyor buffer so that it is not considered as an individual agent and its capacity is not considered either. However, it is considered as a possible location of agent A_5.

The unprocessed leather A_1 can be converted to the back-part shaping leather A_3 at P_1 and then, A_3 placed at buffer BF_3. Also, A_1 can be converted to A_2 at P_2 and then, A_2 is placed at buffer BF_2. Item A_4 can be produced at P_1 using item A_2 or at P_2 using item A_3. Then, A_4 is placed at BF_4. Item A_5 is produced at P_3 using item A_4. Then, A_5 is placed at BF_5. Item A_7 is produced at P_4 using items A_5 and A_6. Finally, item A_7 is placed at BF_7.

4.2 Case Study Results

We run three scenarios utilizing one, two or three workers respectively, so as to study the interaction between workers and buffer zones as the number of workers gradually increases as well as the benefits of utilizing buffer zones in the production line. We assume that there are unlimited resources of unprocessed leather at P_1 and P_2; and soles are at P_4. For the initial conditions we assume that toe-shaping

leather, back-part shaping leather, back-part and toe-shaping leather, lasted leather and shoes have not been produced yet, the workers could be anywhere in the factory.

4.2.1 Scenario I

For the first scenario, we considered eight agents in total; seven items and one worker. The cardinality of the environment's state space is 14,400 states. The objective is to produce A_7 and locate it at BF_7 utilizing one worker.

In the solution computed by SPECTER, the objective is fulfilled after twenty four steps. In words, (T_1) worker goes at work-cell P_1, (T_2) worker inserts A_1 in machine at P_1, (T_3) A_3 is produced at P_1 after 21 seconds, (T_4) worker places A_3 at BF_3, (T_5) worker goes at work-cell P_2, (T_6) worker inserts A_1 in the machine at P_2, (T_7) A_2 is produced at P_2 after 21 seconds, (T_8) worker places A_2 at BF_2, (T_9) worker retrieves A_3 from BF_3 and inserts A_3 in machine at P_2, (T_{10}) A_4 is produced at P_2 after 21 seconds, (T_{11}) worker places A_4 at BF_4, (T_{12}) worker goes at work-cell P_1, (T_{13}) worker retrieves A_2 from BF_2 and inserts A_2 in machine at P_1, (T_{14}) A_4 is produced at P_1 after 21 seconds, (T_{15}) worker places A_4 at BF_4, (T_{16}) worker goes at work-cell P_3, (T_{17}) worker retrieves A_4 from BF_4 and inserts A_4 in machine at P_3, (T_{18}) A_5 is produced at P_3 after 34 seconds, (T_{19}) worker places A_5 at BF_5, (T_{20}) worker goes at work-cell P_4, (T_{21}) worker retrieves A_5 from BF_5 and inserts A_5 in machine at P_4, (T_{22}) worker inserts A_6 in machine at P_4, (T_{23}) A_7 is produced at P_4 after 126 seconds, (T_{24}) worker places A_7 at BF_7. Thus, the task is fulfilled; shoes produced and places at buffer zone BF_7.

The time required to implement the solution is 269 seconds utilizing one worker for the whole process. The diagram of Fig. 4.3 presents the capacity of each buffer zone at every step of the solution of the first scenario. In this case, the capacity of each buffer during the task plan is stated as follows: In the beginning, the buffers are empty. At T_4, the capacity of BF_3 increases from 0 to 1 unit of A_3. At T_8, the capacity of BF_2 increases from 0 to 1 unit of A_2. At T_9, BF_3 capacity decrease from 1 to 0. At T_{11}, BF_4 capacity increase from 0 to 1 unit of A_4. At T_{13}, the capacity of BF_2 decrease from 1 to 0. At T_{15}, the capacity of BF_4 increase from 1 to 2 units of A_4. At T_{17}, the capacity of BF_4 decrease from 2 to 1 unit of A_4. At T_{24}, the capacity of BF_7 decrease from 0 to 1 unit of A_7.

The pre-processing runtime to construct the agents' and environment's models is $3,43 \times 10^3$ seconds. The cardinality of the environment state space is 230,400 states. The runtime to calculate the optimal solution is $22,39$ seconds.

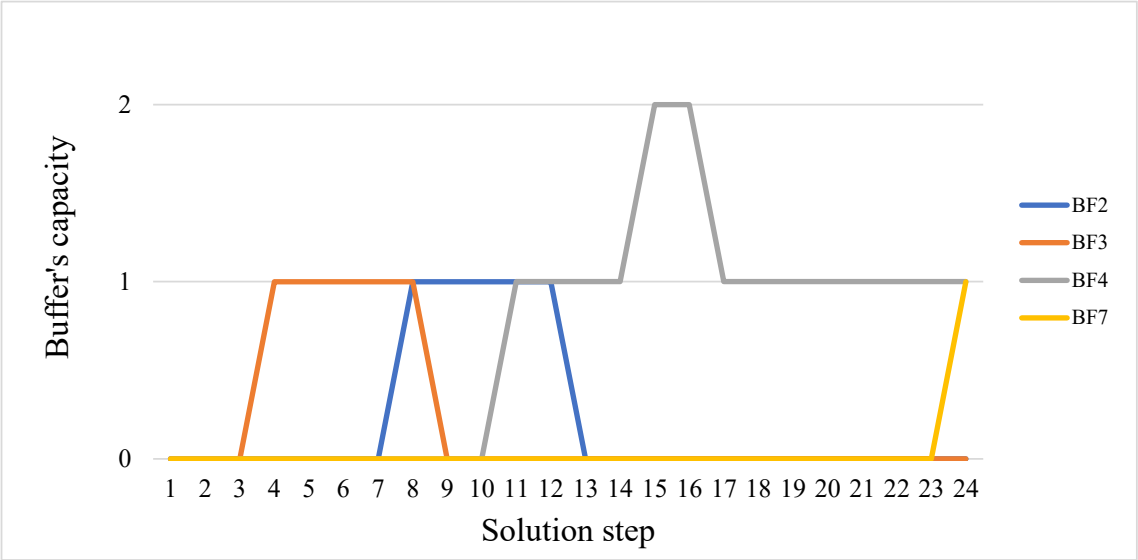


Figure 4.3: Buffers' capacity for the Scenario I.

4.2.2 Scenario II

For the second scenario, we considered nine agents in total; seven items and two workers. The cardinality of the environment's state space is 72,000 states. The objective is to produce A_7 and locate it at BF_7 utilizing two workers.

In the solution computed by SPECTER, the objective is fulfilled after fourteen steps. In words, (T_1) worker A_8 goes at work-cell P_2; worker A_9 goes at work-cell P_1, (T_2) worker A_8 inserts A_1 in machine at P_2, while worker A_9 inserts A_1 in machine at P_1, (T_3) after 21 seconds, A_2 is produced at P_2 and A_3 is produced at P_1, (T_4) worker A_9 places A_3 at BF_3, while worker A_8 places A_2 at BF_2, (T_5) worker A_8 retrieves A_3 from BF_3 and inserts A_3 in the machine at P_2, while worker A_9 retrieves A_2 from BF_2 and inserts A_2 in the machine at P_1, (T_6) two entities of A_4 are produced after 21 seconds; one entity at P_1 and one entity at P_2, (T_7) worker A_9 places A_4 entities at BF_4 while worker A_8 goes at work-cell P_3, (T_8) worker A_8 retrieves A_4 from BF_4 and inserts A_4 in machine at P_3, (T_9) A_5 is produced at P_3 after 34 seconds, (T_{10}) worker A_9 places A_5 at BF_5 while worker A_8 goes at work-cell P_4, (T_{11}) worker A_8 retrieves A_5 from BF_5 and inserts A_5 in machine at P_4, (T_{12}) worker A_8 inserts A_6 in machine at P_4, (T_{13}) A_7 is produced at P_4 after 126 seconds, (T_{14}) worker A_8 places A_7 at BF_7.

The time required to implement the solution with concurrent execution capability requires 238 seconds. The diagram of Fig. 4.4 presents the capacity of each buffer zone at every step of the solution of the second scenario. In this case, the capacity of each buffer during the task plan is as follows: In the beginning, the buffers are empty. At T_4, the capacities of BF_3 and BF_2 increase from 0 to 1 unit of A_3 and A_2, respectively. At T_5, the capacities of BF_3 and BF_2 decrease from 1 to 0, respectively. At T_7, the capacity of BF_4 increase from 0 to 2 units of A_4. At T_8, the capacity of BF_4 decrease from 2 to 1 unit of A_4. At T_{14}, the capacity of BF_7 increase from 0 to 1 unit of A_7.

The pre-processing runtime to construct the agents' and environment's models is $1,72 \times 10^5$ seconds. The cardinality of the environment state space is 1,152,000 states. The runtime to calculate the optimal solution is 647.11 seconds.

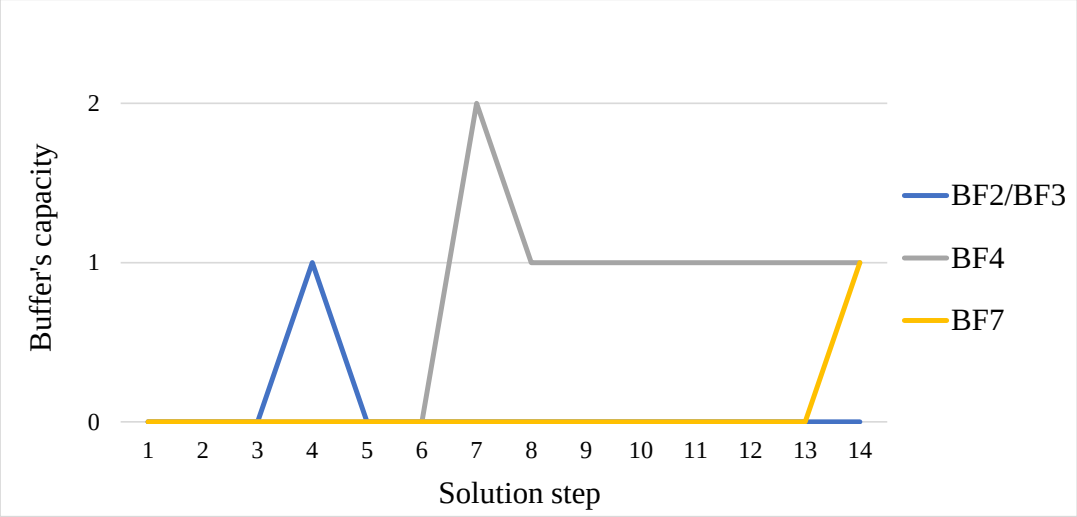


Figure 4.4: Buffers' capacity for the Scenario II. Capacities of BF_2 and BF_3 coincide.

4.2.3 Scenario III

For the third scenario, we considered ten agents in total; seven items and three workers. The cardinality of the environment's state space is 360,000 states. The objective is to produce A_7 and locate it at BF_7 utilizing three workers.

In the solution computed by SPECTER, the objective is fulfilled after fourteen steps. In words, (T_1) worker A_9 goes at work-cell P_2; worker A_{10} goes at work-cell P_1, (T_2) worker A_9 inserts A_1 in machine at P_2, while worker A_{10} inserts A_1 in machine at P_1, (T_3) after 21 seconds, A_2 is produced at P_2 and A_3 is produced at P_1, (T_4) worker A_{10} places A_3 at BF_3, while worker A_9 places A_2 at BF_2, (T_5) worker A_9 retrieves A_3 from BF_3 and inserts A_3 in the machine at P_2, while worker A_{10} retrieves A_2 from BF_2 and inserts A_2 in the machine at P_1, (T_6) two entities of A_4 are produced after 21 seconds; one entity at P_1 and one entity at P_2, (T_7) worker A_{10} places A_4 entities at BF_4 while worker A_9 goes at work-cell P_3, (T_8) worker A_9 retrieves A_4 from BF_4 and inserts A_4 in machine at P_3, (T_9) A_5 is produced at P_3 after 34 seconds, (T_{10}) worker A_{10} places A_5 at BF_5 while worker A_9 goes at work-cell P_4, (T_{11}) worker A_9 retrieves A_5 from BF_5 and inserts A_5 in machine at P_4, (T_{12}) worker A_9 inserts A_6 in machine at P_4, (T_{13}) A_7 is produced at P_4 after 126 seconds, (T_{14}) worker A_9 places A_7 at BF_7.

The time required to implement the solution with concurrent execution capability requires 238 seconds. In this case, the capacity of each buffer during the task plan is the same in the second scenario. The diagram of Fig. 4.4 is used to present also the capacity of each buffer zone at every step of the solution of the third scenario.

The pre-processing runtime to construct the agents' and environment's models is $8,94 \times 10^5$ seconds. The cardinality of the environment state space is 5,760,000 states. The runtime to calculate the optimal solution is $3,43 \times 10^3$ seconds.

4.3 Discussion

The Fig. 4.5 demonstrates the results from the three case studies. As it shown in the chart, interestingly a decrease in the production time was observed only when a second worker was added in the workflow but no change was registered when a third worker was added. The results concluded that an increase above the optimal number of workers will not decrease the production time.

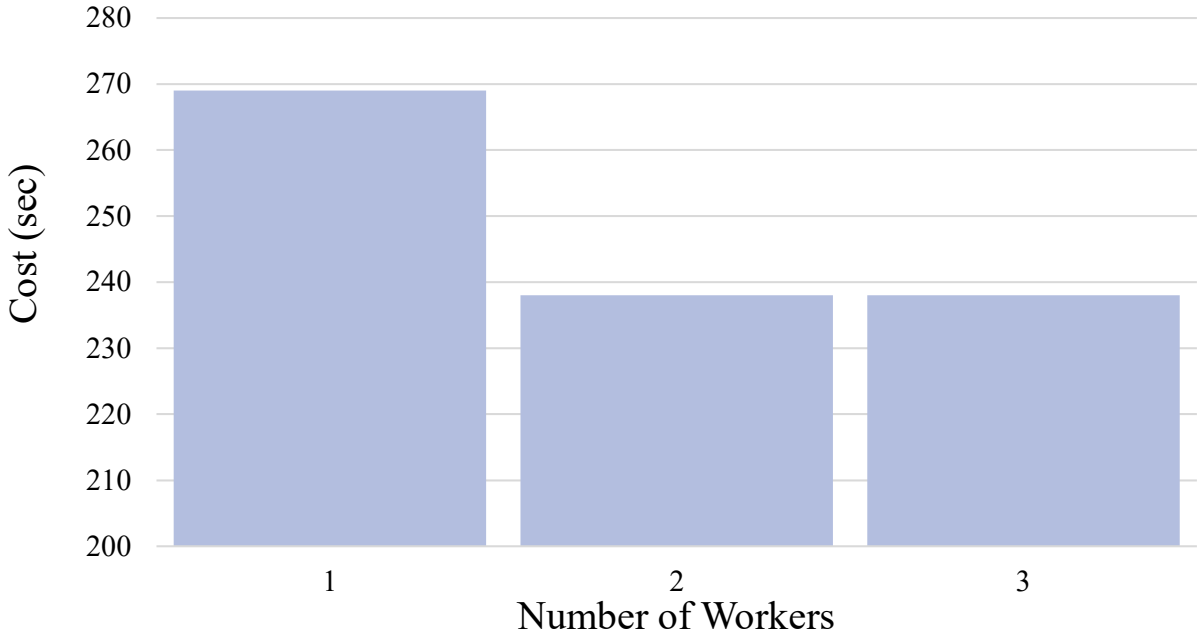


Figure 4.5: Optimal cost (time) for the three case studies.

The buffer zone capacities are shown in Fig. 4.4. For the second and the third scenarios, the parallel execution of tasks are enabled and two units of product A_4 are stored at BF_4 by two workers at step T_7. However, in the first scenario, the worker should repeat the process to store two units of A_4 at BF_4 at step T_{15}. Utilizing the optimal number of workers and retrieving A_4 units from buffer zone BF_4 without the need to re-produce A_4 and repeating the processing steps P_1 and P_2, the overall time of the manufacturing workflow decreases.

We implement three different scenarios to demonstrate the interaction between the buffer zones and the gradual increase of the number of workers involved in the production line, as well as the advantage of utilizing buffer zones in the production line. As shown in Fig. 4.3 and Fig. 4.4, an increase in the capacity of buffer zones is observed, when products are retained so as to be retrieved quickly. This means that the cost of transporting items in the factory is decreased and the overall time of implementing the solution is reduced. Additionally, a decrease in the buffer zone's capacity is observed, when the products are retrieved and utilized to another process.

Similar problems are captured by finite weighted transition systems and finite-time specifications expressed using temporal logic formulas. However, these approaches define the problem domain in providing a solution without considering the planning domain which typically consists of the set of actions along with the pre-conditions and effects. This is not the case in our case, where the problem is defined

within the planning domain. This provides the capability to solve any possible task specification from any possible initial condition utilizing the existing model of the system. On the other hand, sampling-based methods could provide solution to this kind of problem. However, this approach is using a subset of the environment state space to provide a solution by relaxing the properties of completeness and optimality. This means that the state space is not complete, in the sense the possible environment states are not considered while the provided solution is not guaranteed to be the optimal one.

Part III

Supervisory Control

5 Multi-Agent Control Synthesis with Reactive Failure-Mode based Reconfiguration

In this chapter, we are proposing a framework for reactive control synthesis for multi-agent systems, where modeled and unmodeled failures are considered so that the reconfiguration of the task plan satisfies the given task specification (if possible) after the failure occurrence. This concept builds on top of the framework on task planning presented in Chapter 2, where an automatic synthesis framework has been developed to solve the task planning problem for multi-agent systems, leveraging the agents' capabilities and constraints. The framework for automatic task planning under formal language models of Chapter 2 is complemented by a supervisory control framework for handling and/or reconfiguring (in the occurrence of failures) the task plan execution so as to satisfy the given task specification.

Considering that the task planning problem is solved as described in Chapter 2 formulated as a closed module chain, the current work focuses on development of the appropriate supervisory control architecture in the context of robotic motion task planning, for ensuring the satisfaction of the task specification in the occurrence of failures. This provides the capability for an explicit causality relation between the task planner and the agents' controllers. Moreover, the concept of Navigation Transformation [86] under a supervisory control framework provides the required machinery for an analytically guaranteed time-abstracted solution to the motion planning problem.

The main contributions of this chapter are as follows:

- A new automated framework for supervisory control of multi-agent systems under formal specifications handling the task plan execution and the reactive failure reconfiguration.
- A new modeling framework for controller synthesis for agents with hybrid dynamics.
- Capability of setting up the abstraction from the agent's continuous dynamics to appropriate discrete transitions.
- Capability of on-the-fly incorporation of individual and multiple failures.
- Capability of failure categorization based on their direct effect when a failure is known a'priori or not.
- Synthesis of time-abstracting[1] Navigation Transformation controllers[86], under a supervisory control framework.
- Capability of on-the-fly optimal reconfiguration of the task plan to accommodate modeled or unmodeled failures.

The rest of the Chapter is organized as follows: Section 5.1 presents the necessary preliminary notions and definitions while Section 5.2 presents the problem formulation. Section 5.3 describes the proposed methodology and Section 5.4 presents a case study to illustrate the performance of the proposed framework.

[1] Time-abstracting controllers are controllers that provide temporal performance guarantees.

5.1 Preliminaries

5.1.1 Definitions

We will use Definition 1 of Deterministic Finite Automata (DFAs), where the cost function $g_G(e)$ maps each event to a corresponding temporal cost and the concept of Nondeterministic Finite Automata with ϵ transitions (ϵ_0-NFAs) as defined in Definition 2.

Using Corollary 1, the ϵ_0-NFA ${}^{\epsilon_0}G$ can be converted to DFA G by removing x_0 along with the associated ϵ transitions and assigning an $x_{0,G} \in X_G$ as an initial state.

Inspired by the concept of Deterministic Input/Output Automata (DIOA) introduced by [76], we state the following definition in an appropriate form of our development.

Definition 13 (Deterministic Input - Output Automaton)**.** *The deterministic I/O automaton $\mathcal{G}$, modeling the behavior of ${}^{\epsilon_0}G$, is modeled as a 7-tuple of $\mathcal{G} = (\mathcal{Z}_\mathcal{G}, \mathcal{V}_\mathcal{G}, \mathcal{W}_\mathcal{G}, \mathcal{B}_\mathcal{G}, D_\mathcal{G}, H_\mathcal{G}, z_{0,\mathcal{G}})$, where:*

- *$\mathcal{Z}_\mathcal{G}$ is a finite set of states;*
- *$\mathcal{V}_\mathcal{G}$ is the input set;*
- *$\mathcal{W}_\mathcal{G}$ is the output set;*
- *$\mathcal{B}_\mathcal{G}$ is the behavioral relation;*
- *$D_\mathcal{G}$ is the state transition function;*
- *$H_\mathcal{G}$ is the output function;*
- *$z_{0,\mathcal{G}}$ is the initial state;*

such that $\mathcal{B}_\mathcal{G} : \mathcal{V}_\mathcal{G} \to \{0, 1\}$, $D_\mathcal{G} : \mathcal{Z}_\mathcal{G} \times \mathcal{B}_\mathcal{G} \to \mathcal{Z}_\mathcal{G}$, $H_\mathcal{G} : \mathcal{Z}_\mathcal{G} \times \mathcal{V}_\mathcal{G} \times \mathcal{B}_\mathcal{G} \to \mathcal{W}_\mathcal{G}$, $z_{0,\mathcal{G}} \in \mathcal{Z}_\mathcal{G}$.

The deterministic I/O automaton $\mathcal{G}$ is shown in Fig. 5.1. Based on the input that the automaton receives, if the behavioral relation accepts the input v such that $\mathcal{B} = 1$, then an output w is produced and a transition from z to z' is triggered, such that $D(z, 1) = z'$ and $H_\mathcal{G}(z, v, 1) = w'$.

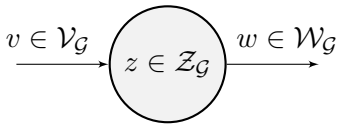


Figure 5.1: Deterministic I/O automaton $\mathcal{G}$.

We introduce the framework of time-abstracting deterministic automata in an appropriate form for our development in order to model and analyze the behavior of ϵ_0-NFAs over a finite set of real-valued clocks.

Definition 14 (Time-Abstracting Deterministic Automaton)**.** *Let ${}^{\epsilon_0}G$ be an ϵ_0-NFA. From Corollary 1, we derive the DFA G. The Time-Abstracting Deterministic Automaton (TADA) $\mathcal{N}$ of DFA G is a tuple of*

$$\mathcal{N} = (\Psi_\mathcal{N}, E_\mathcal{N}, X_G, T_\mathcal{N}, g_G, f_G, \delta_\mathcal{N}, Inv_\mathcal{N}, guard_\mathcal{N}, \rho_\mathcal{N}, \psi_{0,\mathcal{N}}, x_{0,G}, X_{m,G})$$

where:

- *$\Psi_\mathcal{N}$ be a non-empty finite set of discrete modes;*

- *$E_{\mathcal{N}}$ be a non-empty finite set of events;*
- *X_G be the discrete state space of G;*
- *$\varkappa_{\mathcal{N}}$ is a non-empty set of clock variables ($\mathbb{R}_{\geq 0}$);*
- *g_G be a cost function associated to E_G;*
- *f_G be the transition function of G;*
- *$\delta_{\mathcal{N}}$ be the discrete transition function;*
- *$Inv_{\mathcal{N}}$ be an invariant condition;*
- *$guard_{\mathcal{N}}$ be a guard condition;*
- *$\rho_{\mathcal{N}}$ be a reset function;*
- *$\psi_{0,\mathcal{N}}$ be an initial discrete mode, $\psi_{0,\Psi} \in \Psi_{\mathcal{N}}$;*
- *$x_{0,G}$ be the initial discrete state of G, $x_{0,G} \in X_G$;*
- *$X_{m,G}$ be a set of marked states, $X_{m,G} \subseteq X_G$;*

where $E_{\mathcal{N}} = E_G \cup E_{ex}$ with E_{ex} be a set of exogenous events, $g_G : E_G \to \mathbb{R}_{>0}$, $f_G : X_G \times E_G \to 2^{X_G}$, $\delta_{\mathcal{N}} : \Psi_{\mathcal{N}} \times X_G \times E_G \to \Psi_{\mathcal{N}}$, $Inv \subseteq \Psi_{\mathcal{N}} \times X_G$, $guard_{\mathcal{N}} \subseteq \Psi_{\mathcal{N}} \times \Psi_{\mathcal{N}} \times X_G$, $\rho_{\mathcal{N}} : \Psi_{\mathcal{N}} \times \Psi_{\mathcal{N}} \times X_G \times E_G \times \to X_G$.

We will use the definition of hybrid automaton as in [88]. The following definition is in part inspired by [79], but appropriately extended to address the developments in the current work.

Definition 15 (Time-Abstracting Hybrid Automaton)**.** *The Time-Abstracting Hybrid Automaton (TAHA) $\mathcal{Y}$ is a tuple of*

$$\mathcal{Y} = (\mathcal{M}_{\mathcal{Y}}, X_{\mathcal{Y}}, E_{\mathcal{Y}}, U_{\mathcal{Y}}, T_{\mathcal{Y}}, f_{\mathcal{Y}}, \delta_{\mathcal{Y}}, Inv_{\mathcal{Y}}, guard_{\mathcal{Y}}, \rho_{\mathcal{Y}}, q_{0,\mathcal{Y}}, x_{0,\mathcal{Y}}, X_{m,\mathcal{Y}}),$$

consisting of:

- *$\mathcal{M}_{\mathcal{Y}}$ being a non-empty finite set of discrete modes;*
- *$X_{\mathcal{Y}}$ be a set of continuous state space (normally $\mathbb{R}^n$);*
- *$E_{\mathcal{Y}}$ be a non-empty finite set of events;*
- *$U_{\mathcal{Y}}$ be a set of admissible controls (normally $U_{\mathcal{Y}} \subseteq \mathbb{R}^n$);*
- *$\varkappa_{\mathcal{Y}}$ is a non-empty set of clock variables ($\mathbb{R}_{\geq 0}$);*
- *$f_{\mathcal{Y}}$ be a vector field;*
- *$\delta_{\mathcal{Y}}$ be a discrete state transition function;*
- *$Inv_{\mathcal{Y}}$ be an invariant condition;*
- *$guard_{\mathcal{Y}}$ be a guard condition;*
- *$\rho_{\mathcal{Y}}$ be a reset function;*
- *$m_{0,\mathcal{Y}}$ be an initial discrete state, $m_{0,\mathcal{Y}} \in \mathcal{M}_{\mathcal{Y}}$;*
- *$x_{0,\mathcal{Y}}$ an initial continuous state, $x_{0,\mathcal{Y}} \in X_{\mathcal{Y}}$;*

- *$X_{m,\mathcal{Y}}$ be a finite set of marked states, $X_{m,\mathcal{Y}} \subseteq X_{\mathcal{Y}}$;*

where $f_{\mathcal{Y}} : \mathcal{M}_{\mathcal{Y}} \times X_{\mathcal{Y}} \times U_{\mathcal{Y}} \to X_{\mathcal{Y}}$, $\delta_{\mathcal{Y}} : \mathcal{M}_{\mathcal{Y}} \times X_{\mathcal{Y}} \times E_{\mathcal{Y}} \to \mathcal{M}_{\mathcal{Y}}$, $Inv_{\mathcal{Y}} \subseteq \mathcal{M}_{\mathcal{Y}} \times X_{\mathcal{Y}}$, $guard_{\mathcal{Y}} \subseteq \mathcal{M}_{\mathcal{Y}} \times \mathcal{M}_{\mathcal{Y}} \times X_{\mathcal{Y}}$, $\rho_{\mathcal{Y}} : \mathcal{M}_{\mathcal{Y}} \times \mathcal{M}_{\mathcal{Y}} \times X_{\mathcal{Y}} \times E_{\mathcal{Y}} \to X_{\mathcal{Y}}$.

5.2 Problem Formulation

5.2.1 Agents Modeling

In the proposed framework, two classes of agents are considered. Let n be the number of agents. Let us consider the set N_c of agents with continuous dynamics and the set N_d of agents with purely discrete dynamics, such that $n = |N_c| + |N_d|$. Moreover, let us assume that both agent sets can be abstracted to ϵ_0-NFAs agent models. The restriction of the considered agents between continuous and discrete does not reduce the applicability of the methodology. In practice any agent that can be abstracted to an ϵ_0-NFA can be considered. The i^{th} agent is denoted by A_i and modeled by the ϵ_0-NFA ${}^{\epsilon_0}A_i$, where $i \in \{1, \ldots, |N_c|, |N_c| + 1, \ldots, n\}$. Let us now consider how each agent category can be appropriately abstracted to ϵ_0-NFAs.

5.2.1.1 Agents with Discrete Dynamics

The first class of agents that we are considering is the agents with purely discrete dynamics modeled as ϵ_0-NFAs. Following the procedure described in Chapter 2, these can be converted to ϵ_0-NFAs representations. Thus, the j'th agent $j \in \{|N_c| + 1, \ldots, n\}$ is modeled as ${}^{\epsilon_0}A_j \triangleq (X_{A_j} \cup \{x_0\}, E_{A_j} \cup \{\epsilon\}, {}^{\epsilon_0}f_{A_j}, \Gamma_{A_j}, x_0, X_{m,A_j})$. The cost function associated with the original DFA transitions of each agent is represented as g_{A_j}. The system dynamics are captured by the state transition equation:

$$\alpha'_j = f_{A_j}(\alpha_j, e_j) \tag{5.1}$$

where $\alpha_j, \alpha'_j \in X_{A_j}, e_j \in E_{A_j}$.

5.2.1.2 Agents with Continuous Dynamics

The second class of agents that we are considering is the agents with hybrid dynamics, that are agents with both discrete and continuous dynamics. Those agents can be expressed as ϵ_0-NFAs. Thus, the discrete dynamics of an agent are captured by the state transition function of the relevant ϵ_0-NFA as in eq. 5.1. In the current work, we will restrict our focus to agents represented by robots with continuous dynamics utilizing the Navigation Transformation concept [86] to satisfy motion task plans. Assume that each robotic agent's dynamics are described by the following equation:

$$\dot{x}_k = h_k(x_k, u_k), \quad k \in \{1, \ldots, |N_c|\} \tag{5.2}$$

where $x_k \in \mathbb{R}^\mu$ is the μ-dimensional configuration space of the robot agent A_k and u_k is the control input. Hence, the continuous dynamics of each agent are captured by the vector field h_k which depends on the continuous state X_k and the control input u_k of each agent.

Let ϕ and e denote the state and event that will be used for setting up the abstraction from the robot's continuous dynamics to appropriate discrete transitions. Let us assume a partitioning of the robots' configuration space to L partitions corresponding to areas of interest. In each area of interest we assign a reference configuration $x_\phi,\ \phi \in \{1, \ldots, L\}$. For every reference configuration x_ϕ, we assign an event $e_{\phi,\phi'}$, denoting the capability of the system to navigate to reference configuration $x'_\phi,\ \phi' \neq \phi$ within time $T_{\phi,\phi'}$ using the Navigation Transformation based controller. We map each couple e, ϕ to a destination reference configuration ϕ' and assign to event e a cost $g_{A_k}(e) = T_{\phi,\phi'}$. The transition $f_{A_k}(\phi, e) = \phi'$ then corresponds to an *Individual Capability* (see Section 2.2.2.1). Furthermore, we can identify the following cases for navigating from area of interest ϕ to area ϕ' that can be modeled under the proposed framework:

(i) Agents are forbidden or restricted to navigate to or from areas of interest. Then, for each restriction, we construct the corresponding *Individual Constraint* (see Section 2.2.2.4).

(ii) Navigation between selected areas of interest are prone to failures. Then, for each failure occurrence, we construct a corresponding *Individual Failure Mode* (see Section 2.2.2.2).

(iii) Navigation between selected areas of interest is conditionally enabled by other agent(s). Then, for each such navigation scenario we construct the corresponding *Inter-Agent Capability* (see Section 2.2.3.3.

(iv) Navigation between selected areas of interest is conditionally disabled by other agent(s). Then, for each such navigation scenario we construct the corresponding *Inter-Agent Constraint* (see Section 2.2.3.4).

Following this procedure, we can map all the navigation capabilities, failure modes and constraints of an agent and construct the ϵ_0-NFA agent model ${}^{\epsilon_0}A_k$. Note that the individual and inter-agent capabilities and constraints as well as failure modes can be defined in a similar fashion for the agents with discrete dynamics (where appropriate), whereas inter-agent capabilities and constraints can be defined between all agents (with discrete or/and continuous dynamics) that have been modeled as ϵ_0-NFAs.

The Navigation-Transformation-based time-abstracting controllers are constructed as in Proposition 3 in [86]. This controller ensures that the navigation from area on interest q to q' related to the transition $f(q, e)$ is achieved within time $T_k = g(e)$ (assuming that this is physically possible). Thus, the control input u_k is parameterized though the discrete state q and the event e and depends on the continuous state x_k and the specific time t_k for navigation. Hence, the controller is of the form:

$$u_k = u_{q,e}(x_k, t_k), \quad k \in \{1, \ldots, |N_c|\}. \tag{5.3}$$

Therefore, each navigation task between two areas of interest can be abstracted to an event triggering the corresponding Navigation Transformation based controller and the corresponding cost is the time allocated for the completion of the task. Consequently, depending on the discrete state q and the event e, the control input u_k is parameterized with respect to its destination configuration x_d and the specific time T_k to execute the task.

5.2.2 Types of Failures

As part of the proposed framework for modeling the agent's behavior, we consider two categorizations of failures that contribute to an infeasible task execution of an agent.

In the first categorization, failures are categorized based on their direct effect, i.e. the *individual* failure, when the failure directly affects the individual agent's capabilities and the *environmental* failure, when the failure directly affects capabilities of the environment and not just of an individual agent. Individual agents' failures may indirectly affect the environment (other agents). The environmental failures are collective failures that can be encoded as inter-agent constraints in our framework.

In the second categorization, especially important when considering systems implementing fault diagnosers, we consider *modeled* and *unmodeled* failures. Modeled failures are well defined individual and environmental failures that are known *a priori*. Unmodeled failures are failure modes that will go unnoticed until a supervisor detects that a task is not performing up to its specifications. In the current work we will consider unmodeled failures that cause tasks to exceed their allocated temporal cost. Individual and environmental failures can be either modeled or unmodeled. We will need the following definition:

Definition 16 (Environmental Failure Mode)**.** *Consider the event $e \in E_{\mathcal{S}}$ that triggers a transition from x_α to x_d such that $f_{\mathcal{S}}(x_\alpha, e) = x_d$, where $x_\alpha, x_d \in X_{\mathcal{S}}$. Let b_i be the n-bit binary number that indicates the i'th element of x, which corresponds to the status of agent A_i. We define the environmental failure mode that renders $f_{\mathcal{S}}(x_\alpha, e) \neq x_d$, representing a detected transition failure, modeled as*

$$^{\epsilon_0}\mathcal{F}_{\mathcal{S}} = (X_{\mathcal{F}_{\mathcal{S}}} \cup \{x_0\}, E_{\mathcal{F}_{\mathcal{S}}} \cup \{\epsilon\}, {}^{\epsilon_0}f_{\mathcal{F}_{\mathcal{S}}}, \Gamma_{\mathcal{F}_{\mathcal{S}}}(e), x_0, X_{m,\mathcal{F}_{\mathcal{S}}}),$$

where

$$X_{\mathcal{F}_{\mathcal{S}}} = \{x_\beta |\, proj(x_\beta, b_i) \in \{proj(x_\alpha, b_i), proj(x_d, b_i)\},$$

$$E_{\mathcal{F}_{\mathcal{S}}} = \{e_\beta |\, \exists x_\beta \in X_{\mathcal{F}_{\mathcal{S}}} : proj(f_{\mathcal{S}}(x_\beta, e_\beta), b_i) = proj(f_{\mathcal{S}}(x_\alpha, e_\beta), b_i),$$

$f_{\mathcal{F}_{\mathcal{S}}} : X_{\mathcal{F}_{\mathcal{S}}} \times E_{\mathcal{F}_{\mathcal{S}}} \to X_{\mathcal{F}_{\mathcal{S}}}$, $\Gamma_{\mathcal{F}_{\mathcal{S}}} : X_{\mathcal{F}_{\mathcal{S}}} \to 2^{E_{\mathcal{F}_{\mathcal{S}}}}$, $X_{m,\mathcal{F}_{\mathcal{S}}} \subseteq X_{\mathcal{F}_{\mathcal{S}}}$.

Using Definition 5, it holds that $^{\epsilon_0}\mathcal{S}$ and $^{\epsilon_0}\mathcal{F}_{\mathcal{S}}$ are compatible, denoted as $^{\epsilon_0}\mathcal{S} \asymp {}^{\epsilon_0}\mathcal{F}_{\mathcal{S}}$. Considering $^{\epsilon_0}\mathcal{S} \asymp {}^{\epsilon_0}\mathcal{F}_{\mathcal{S}}$, then $^{\epsilon_0}\mathcal{F}_{\mathcal{S}}$ is incorporated in $^{\epsilon_0}\mathcal{S}$ using the operation of subtraction on compatible automata as in Definition 7.

Proposition 3. *Let $^{\epsilon_0}\mathcal{S}$ be an environment model as in [95] and $^{\epsilon_0}\mathcal{F}_{\mathcal{S}}$ be an environmental failure mode as in Definition 16. Then, $^{\epsilon_0}\mathcal{S}' := {}^{\epsilon_0}\mathcal{S} \setminus_{\asymp} {}^{\epsilon_0}\mathcal{F}_{\mathcal{S}}$ is an environment model. Then, the following are true:*

(a) $^{\epsilon_0}\mathcal{S}'$ is complete.

(b) Given a task specification γ, the $^{\epsilon_0}\mathcal{S}'$ and the cost function $g_{\mathcal{S}'}$, the optimal solution (if such exists) will be represented by a run[2] in $^{\epsilon_0}\mathcal{S}'$.

Proof. 1. Following the approach of Proposition 1, the property of completeness is maintained through the subtraction operation. Hence, all and only valid transitions are present in the event set

[2] Automaton run, as defined in [88].

of ${}^{\epsilon_0}\mathcal{S}$. In particular in the case of an Environmental Failure Mode occurrence, the event set of $E_{\mathcal{F}_\mathcal{S}}$ is removed from the global capabilities of the environment model without affecting events that are not included in the global capabilities event set.

2. Based on the Proposition 2 and since ${}^{\epsilon_0}\mathcal{S}'$ is complete, then all possible solutions are encoded. Hence, the optimal solution will also be encoded.

□

Based on Definition 16, we can proceed with the following construction. Considering an event $e \in E_\mathcal{S}$ and the string π of cardinality n, containing the agents' execution status and the event that caused a failure (if any) on each agent. Then, define the following operator:

$$\Omega(\pi) \triangleq {}^{\epsilon_0}\mathcal{F}_\mathcal{S}. \tag{5.4}$$

Operator $\Omega(\pi)$, given the fault state of the system, synthesizes an Environmental Failure Mode as in Definition 16.

Note that using eq. (5.4), and since any individual failure mode automaton ${}^{\epsilon_0}F_i$ can be expressed as an appropriate couple of $proj(\pi, b_i) \triangleq e_i \in {}^{\epsilon_0}F_i$, then any ${}^{\epsilon_0}F_i$ can be converted to an environmental failure mode automaton ${}^{\epsilon_0}\mathcal{F}_\mathcal{S}$ using the Ω operator.

5.2.3 Problem Statement

Assume a multi-agent system consisting of N_d agents with purely discrete dynamics interacting with N_c agents with continuous dynamics. All agents are equipped with time-abstracting controllers. Further assume that each agent has a local diagnoser that can identify a set of modeled failures but that unmodeled failures can occur.

The problem statement can then be stated as follows:

Given a task specification defining the final state of at least one agent, synthesize and reconfigure in the case of failures, the appropriate control actions in terms of events and control inputs for eq. (5.1) and eq. (5.2) respectively, so that the multi-agent system can satisfy the task specification as long as a solution exists.

5.3 Our Approach

5.3.1 Time-Abstracting Deterministic Automata with Failure Modes for Agents with Discrete Dynamics

Using Definition 14, we proceed to model the behavior of agents with purely discrete dynamics incorporating modeled and unmodeled failures. The TADA $\mathcal{N}_j$ of agent A_j is illustrated in Fig. 5.2. The discrete state set $\Psi_{\mathcal{N}_j} = \{0, 1, 2, 3\}$ represents the status of A_j at a time instant with the state "0" being the idle state, the state "1" being the normal execution state, the failure state "2" representing the occurrence of unmodeled failure(s) and the failure state "3" representing the occurrence of modeled failure(s).

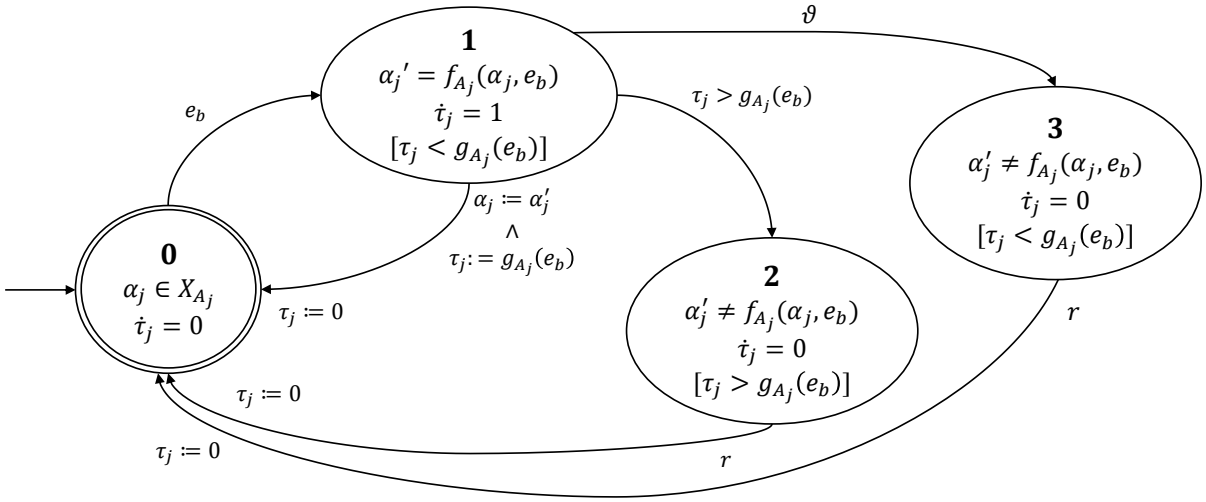


Figure 5.2: The $\mathcal{N}_j$ TADA of agent A_j with discrete dynamics and failure modes.

Let $e_j \in E_{\mathcal{N}_j}$ and $\tau_j(t) \in T_{\mathcal{N}_j}$ be a clock variable for timeouts, $\psi \in \Psi_{\mathcal{N}_j}$ and $E_{\mathcal{N}_j} = E_{A_j} \cup E_{ex}$, where $E_{ex} = \{\vartheta, r\}$. The α_j is the actual state of the monitored system, whereas α_j' is the predicted state that the system must transition to. In the beginning, the automaton is on its starting state which means that the agent is not activated yet and the clock is not started. Thus, for $\psi = 0$, α_j and $\tau_j(t)$ remain unchanged. A transition from state "0" to state "1" can only occur when an event $e_b \in E_{A_j}$ is activated to initiate the normal execution. Thus, the discrete transition function is defined at state $\psi = 0$ as:

$$\delta_{\mathcal{N}_j}(\psi = 0, \alpha_j, e_j) = \begin{cases} 1, & \text{if } e_j := e_b \in E_{A_j} \\ 0 & \text{otherwise.} \end{cases}$$

When in $\psi = 1$, the clock is started with $\dot{\tau}_j = 1$, agent's A_j discrete dynamics evolve according to eq. (5.1). The invariant condition $\tau_j < g_{A_j}(e_b)$ is satisfied as long as the system remains in $\psi = 1$. Once, the destination state α_j' is reached and the time reaches the assigned time, such that $\tau_j = g_{A_j}(e_b)$, the guard condition determines that the operation is finished at time $g_{A_j}(e_b)$ indicating a normal execution. Thus, the transition from state "1" to state "0" is occurred if $\alpha_j{}'$ is reached until $\tau_j = g_{A_j}(e_b)$. Then, $\alpha_j := \alpha_j'$ and the clock is reset, $\tau_j := 0$, while $\mathcal{N}_j$ returns to state "0" waiting for an event to start its normal execution. In case of the clock exceeds the assigned time, such that $\tau_j > g_{A_j}(e_b)$, a transition from state "1" to state "2" is forced, denoting that an unmodeled failure has occurred for agent A_j. An occurrence of the exogenous event ϑ, originating from the fault diagnoser and denoting the occurrence of a modeled failure, forces an immediate transition from state "1" to state "3" while the invariant condition is $\tau_j < g_{A_j}(e_b)$. Thus,

$$\delta_{\mathcal{N}_j}(\psi = 1, \alpha_j, e_j) = \begin{cases} 0, & \text{if } \alpha_j := \alpha_j',\ \tau_j := g_{A_j}(e_b) \\ 2, & \text{if } \tau_j > g_{A_j}(e_b) \\ 3, & \text{if } e_j := \vartheta \\ 1 & \text{otherwise.} \end{cases}$$

In state "2", the clock stops while $\mathcal{N}_j$ is set back to state "0" with the reset condition $\tau_j := 0$, $\alpha_j := \alpha'_j$. Thus,

$$\delta_{\mathcal{N}_j}(\psi = 2, \alpha_j, e_j) = \begin{cases} 0, & \text{if } e_j := r \\ 2 & \text{otherwise.} \end{cases}$$

In state "3", the clock stops and the $\mathcal{N}_j$ is set back to state "0" with the reset condition $\tau_j := 0$, $\alpha_j := \alpha'_j$. Thus,

$$\delta_{\mathcal{N}_j}(\psi = 3, \alpha_j, e_j) = \begin{cases} 0, & \text{if } e_j := r \\ 3 & \text{otherwise.} \end{cases}$$

5.3.2 Time-Abstracting Hybrid Automata with Failure Modes for Agents with Continuous Dynamics

Using Definition 15, we proceed to model the behavior of agents with continuous dynamics incorporating modeled and unmodeled failures. The TAHA $\mathcal{Y}_k$ of agent A_k is illustrated in Fig. 5.3. The discrete state set $\mathcal{M}_k = \{0, 1, 2, 3\}$ represents the status of A_k at a time instant with the state "0" being the idle state, the state "1" being the normal execution state, the failure state "2" presenting the occurrence of unmodeled failure(s) and the failure state "3" presenting the occurrence of modeled failure(s).

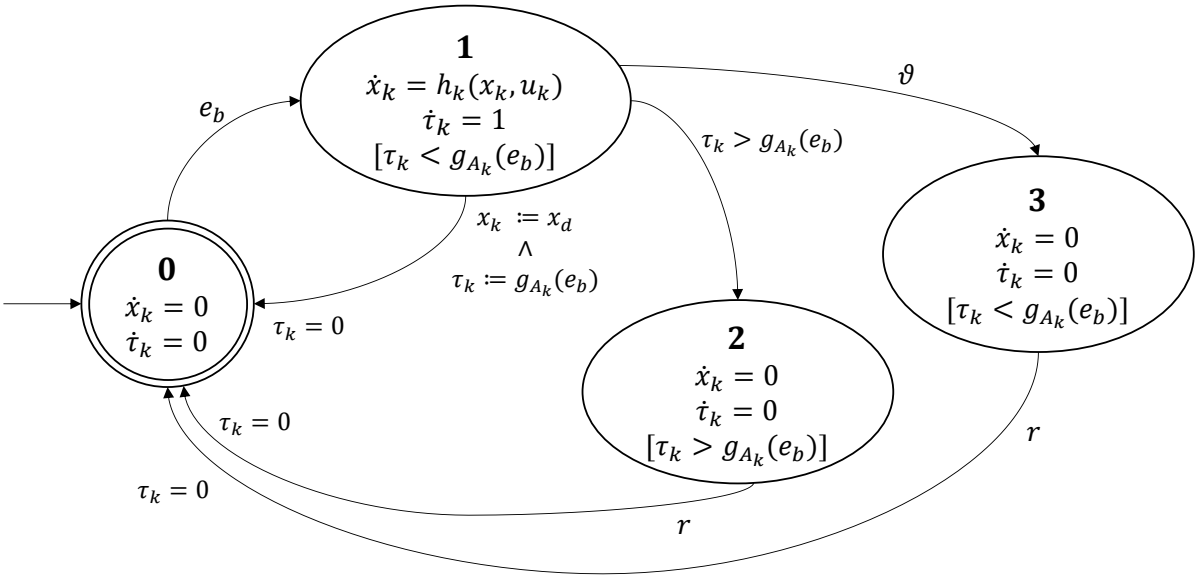


Figure 5.3: The $\mathcal{Y}_k$ TAHA of agent A_k with continuous dynamics and failure modes.

Let $x_k(t)$ be the continuous state variable of the monitored system corresponding to agent A_k and, $\tau_k(t)$ the clock variable for timeouts, $m_k \in \mathcal{M}_k$ and $E_{\mathcal{Y}_k} = E_{A_k} \cup E_{ex}$, where $E_{ex} = \{\vartheta, r\}$ (as in TADA), and $e_k \in E_{\mathcal{Y}_k}$, $e_b \in E_{A_k}$. In the beginning, the automaton is on its starting state which means that the agent is not moving yet and the clock is not activated. Thus, for $m_k = 0$, then $x_k(t)$ and $\tau_k(t)$ remain unchanged. A transition from state "0" to state "1" can only occur when an event $e_b \in E_{A_k}$ is activated

to initiate the normal execution. Thus, the discrete transition function is defined as:

$$\delta_{\mathcal{Y}_k}(m_k = 0, x_k, e_k) = \begin{cases} 1, & \text{if } e_k := e_b \in E_{A_k} \\ 0 & \text{otherwise.} \end{cases}$$

When in state $m_k = 1$, the clock is started with $\dot{\tau}_k = 1$, agent's A_k dynamics evolve according eq. (5.2), where the evolution of the continuous state of the system is dictated by the vector field h_k which is parameterized by the state x_k and the control u_k is parameterized with respect to the destination configuration x_d and the duration $g_{\mathcal{A}_k}(e_b)$ such that $u_k = u_{\phi,e}(x_d, g_{\mathcal{A}_k}(e_b))$. The invariant condition $\tau_k < g_{\mathcal{A}_k}(e_b)$ is satisfied as long as the system remains in state "1". Once the destination configuration x_d is reached and the time reaches the assigned time, such that $\tau_k = g_{\mathcal{A}_k}(e_b)$, the guard condition determines that the action is finished at time τ_k indicating a normal execution. The navigation task completion at the specific time instant is guaranteed by the underlying time-abstracting controllers $u_{\phi,e}$. Hence, the automaton transitions from state "1" to state "0" if $\tau_k := g_{\mathcal{A}_k}(e_b)$ and $x_k := x_{k,d}$, waiting for an event to start its normal execution, while the reset condition is $x_k = 0$ and the clock is reset, $\tau_k = 0$. In case the clock exceeds the assigned time, such that $\tau_k > g_{\mathcal{A}_k}(e_b)$, a transition from state "1" to state $m_k = 2$ is forced, denoting that an unmodeled failure has occurred for agent A_k. An occurrence of the exogenous event ϑ, originating from the fault diagnoser and denoting the occurrence of a modeled failure, forces an immediate transition from state "1" to state "3" while the invariant condition is $\tau_k < g_{\mathcal{A}_k}(e_b)$. Thus, the transition function in state $m_k = 1$ is defined as:

$$\delta_{\mathcal{Y}_k}(m_k = 1, x_k, e_k) = \begin{cases} 0, & \text{if } x_k := x_d,\ \tau_k := g_{\mathcal{A}_k}(e_b) \\ 2, & \tau_k > g_{\mathcal{A}_k}(e_b) \\ 3, & \text{if } e_k := \vartheta \\ 1 & \text{otherwise.} \end{cases}$$

In state $m_k = 2$, the reset event r is activated while the clock is stopped. The automaton state is set back to state "0", where the reset condition is $\tau_k = 0$. Thus,

$$\delta_{\mathcal{Y}_k}(m_k = 2, x_k, e_k) = \begin{cases} 0, & \text{if } e_k := r \\ 2 & \text{otherwise.} \end{cases}$$

In $m_k = 3$, r is activated and the automaton is set back to state "0". Thus,

$$\delta_{\mathcal{Y}_k}(m_k = 3, x_k, e_k) = \begin{cases} 0, & \text{if } e_k := r \\ 3 & \text{otherwise.} \end{cases}$$

5.3.3 Supervisory Control Architecture

Using Definition 13, we proceed to model the global supervisor and local supervisor of each agent as follows:

5.3.3.1 Global Supervisor

The global supervisor $\mathcal{C}$ is modeled by the deterministic I/O automaton $\mathcal{C} = (\mathcal{Z}_\mathcal{C}, \mathcal{V}_\mathcal{C}, \mathcal{W}_\mathcal{C}, \mathcal{B}_\mathcal{C}, D_\mathcal{C}, H_\mathcal{C}, z_{0,\mathcal{C}})$, where:

- $\mathcal{Z}_\mathcal{C} = \{1, \ldots, |\mathcal{T}_{\text{max}}|\}$,
- $\mathcal{V}_\mathcal{C} = (\mathcal{T}, \pi)$,
- $\mathcal{W}_\mathcal{C} = \{e_b, (e_f, \pi)\}$ with $e_b \in T_{z_C}$, $e_f \in \{T_1, \ldots, T_{z_C}\}$,
- $\mathcal{B}_\mathcal{C}(\pi) = \{0, 1\}$,
- $D_\mathcal{C}(z_C, \mathcal{B}_\mathcal{C}(\pi)) = z_C + 1$,
- $H_\mathcal{C}(z_C, \mathcal{B}_\mathcal{C}(\pi)) = \{e_b(z_C), (e_f(z_C), \pi)\}$
- $z_{0,\mathcal{C}} := 1$

where $\mathcal{T} = \{T_1, \ldots, T_n\}$, z_C is the current state and $|\mathcal{T}_{\text{max}}|$ an upper bound on the module chain size.

5.3.3.2 Local Supervisor of agent with discrete dynamics

The local supervisor $\mathcal{L}_j$ controlling agent $j \in N_d$ is modeled by the deterministic I/O automaton

$$\mathcal{L}_j = (\mathcal{Z}_{\mathcal{L}_j}, \mathcal{V}_{\mathcal{L}_j}, \mathcal{W}_{\mathcal{L}_j}, \mathcal{B}_{\mathcal{L}_j}, D_{\mathcal{L}_j}, H_{\mathcal{L}_j}, z_{0,\mathcal{L}_j}),$$

where

- $\mathcal{Z}_{\mathcal{L}_j} = \Psi_{\mathcal{N}_j}$,
- $\mathcal{V}_{\mathcal{L}_j} = (e_b, \psi)$ with $e_b \in E_{A_j},\ \psi \in \Psi_{\mathcal{N}_j}$,
- $\mathcal{W}_{\mathcal{L}_j} = (\psi, e_j)$ with $e_j \in E_{\mathcal{N}_j}$,
- $\mathcal{B}_{\mathcal{L}_j}(e_b) = \{0, 1\}$,
- $D_{\mathcal{L}_j}(z_{\mathcal{L}_j}, \mathcal{B}_{\mathcal{L}_j}(e_b)) = \delta_{\mathcal{N}_j}(z_{\mathcal{L}_j}, \cdot)$,
- $H_{\mathcal{L}_j}\mathcal{L}_j z_{\mathcal{L}_j}, \mathcal{B}_{\mathcal{L}_j}(e_b)) = \{\psi', (e_b, e_f)\}$,
- $z_{0,\mathcal{L}_j} := 0 \in \Psi_{\mathcal{N}_j}$.

5.3.3.3 Local Supervisor of agent with continuous dynamics

The local supervisor $\mathcal{L}_k$ controlling agent $k \in N_c$ is modeled by the deterministic I/O automaton

$$\mathcal{L}_k = (\mathcal{Z}_{\mathcal{L}_k}, \mathcal{V}_{\mathcal{L}_k}, \mathcal{W}_{\mathcal{L}_k}, \mathcal{B}_{\mathcal{L}_k}, D_{\mathcal{L}_k}, H_{\mathcal{L}_k}, z_{0,\mathcal{L}_k}),$$

where

- $\mathcal{Z}_{\mathcal{L}_k} = \mathcal{M}_{\mathcal{Y}_k}$,
- $\mathcal{V}_{\mathcal{L}_k} = (e_b, m_k)$ with $e_b \in E_{\mathcal{Y}_k},\ m_k \in \mathcal{M}_{\mathcal{Y}_k}$,
- $\mathcal{W}_{\mathcal{L}_k} = \{m_k, (u_k, e_f)\}$ with $u_k \in U_k$, $e_f \in E_{\mathcal{Y}_k}$,
- $\mathcal{B}_{\mathcal{L}_k}(e_b) = \{0, 1\}$,

- $D_{\mathcal{L}_k}(m_k, \mathcal{B}_{\mathcal{L}_k}(e_b)) = m'_k$,
- $H_{\mathcal{L}_k}(m'_k, \mathcal{B}_{\mathcal{L}_k}(e_b)) = \{m'_k, (u_k, e_f)\}$,
- $z_{0,\mathcal{L}_k} := 0 \in \mathcal{M}_{\mathcal{Y}_k}$.

The control synthesis architecture is presented in Fig. 5.4. Given a task specification γ, the SPECTER task planner calculates the task plan $\mathcal{T}$ which is provided to the global supervisory controller $\mathcal{C}$. The global supervisor $\mathcal{C}$ interacts with agents' local supervisors $\mathcal{L}_i$. The global supervisor $\mathcal{C}$ decomposes the task plan and starts its process with the module $T_b := T_1$ of the module chain. The event $e_b := e_1$ is broadcast to the local supervisors $\mathcal{L}_i$. Each local supervisor $\mathcal{L}_i$, interacts with TADAs and TAHAs assigned to agents based on the type of agent dynamics in order to control and supervise the task execution of the respective agents. TAHAs and TADAs are initialized with the initial state of their corresponding agents.

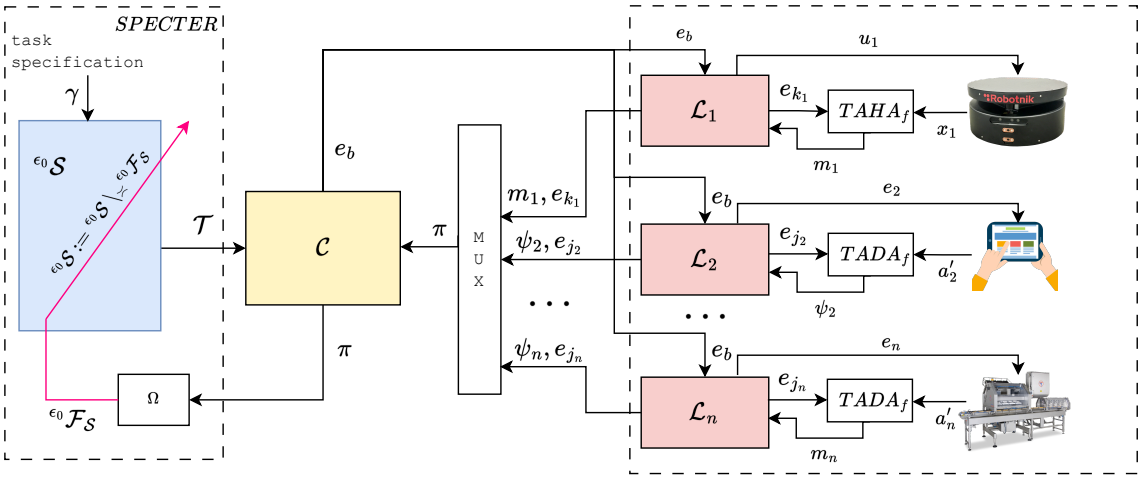


Figure 5.4: Supervisory control architecture. $\mathcal{C}$ is the global supervisor whereas $\mathcal{L}_i$ denote the local supervisors.

The local supervisor $\mathcal{L}_j$ of agent A_j with discrete dynamics, interacting with agent's TADA, publishes the event $e_j := e_b$ to A_j. If the TADA recognizes the received event, i.e. $e_j \in E_{A_j}$, then a transition occurs from TADA state "0" to TADA state "1", and the agent starts the task execution according to the agent's discrete dynamics (eq. (5.1)). In this state, TADA's clock measures the time until the agent has reached the state predicted by eq. (5.1). The TADA receives the current discrete state α'_j of agent A_j and reacts according to $\delta_{\mathcal{N}_j}(\psi = 1, \alpha_j, e_j)$. Based on the received α'_j, the TADA publishes the discrete agent mode/status ψ_j to the $\mathcal{L}_j$.

The local supervisor $\mathcal{L}_k$ of agent A_k with hybrid dynamics, interacting with agent's TAHA, publishes the event $e_j := e_b$ to A_k. If the TAHA recognizes the received event, i.e. $e_k \in E_{A_k}$, then a transition occurs from TAHA state "0" to TAHA state "1", and the agent starts the task execution according to the agent's dynamics (eq. (5.2)). In this state, TAHA's clock measures the time until the agent has reached the state predicted by eq. (5.2). The TAHA receives the current configuration state x_k from A_k as described in section 5.3.2. The TAHA then reacts according to $\delta_{\mathcal{Y}_k}(m_k = 1, x_k, e_k)$. Based on the received x_k, the TAHA publishes the discrete agent mode/status m_k to the $\mathcal{L}_k$.

Thus, ψ_j and m_k report to $\mathcal{L}_j$ and $\mathcal{L}_k$ accordingly the agent status of A_j and A_k whether the execution was successful or whether a failure was detected and its type. The local controllers $\mathcal{L}_i$ publish the ψ_j and m_k to

the multiplexer MUX, which collects the execution status of agents (discrete modes of TADAs/TAHAs) and the failed event (if any failure is occurred) to synthesize the environmental status projector π as an n-string.

We will now proceed with defining the operation of the MUX module. Let $\xi_i := \psi_j, \forall i \in N_d$ and $\xi_i := m_k, \forall i \in N_c$. The set $\Xi \triangleq \{\xi_1, \ldots, \xi_n\}$. Then, the output of the MUX module is defined as:

$$\text{MUX}(\Xi) \triangleq \pi,$$

In particular, it holds that:

$$\xi_i = proj(\pi, b_i),\ \forall i \in \{1, \ldots, n\},$$

where b_i is defined as in Definition 16.

If no failure is detected, then $\xi_i \in \{0, 1\}$. If we have that $\forall i \in \{1, \ldots, n\}$, $\xi_i = 0$, then the supervisor $\mathcal{C}$ proceeds with the next module T_{b+1}. The process repeats itself as long as $b < |\mathcal{T}|$ or until a failure is detected on at least one agent (see below).

In case that an unmodeled failure is detected on the discrete agent A_j, i.e. $\psi_j = 2$, then $\xi_i := e_b$, while the TADA reacts according to $\delta_{\mathcal{N}_j}(\psi_j = 2, \alpha_j, e_j)$. In case that a modeled failure is detected, i.e. $\psi_j = 3$, then $\xi_i := \vartheta$, while the TADA reacts according to $\delta_{\mathcal{N}_j}(\psi_j = 3, \alpha_j, e_j)$.

In case that an unmodeled failure is detected on the hybrid agent A_k, i.e. $m_k = 2$, then $\xi_i := e_b$, while the TAHA reacts according to $\delta_{\mathcal{Y}_k}(m_k = 2, x_k, e_k)$. In case that a modeled failure is detected, i.e. $m_k = 3$, then $\xi_i := \vartheta$, while the TAHA reacts according to $\delta_{\mathcal{Y}_k}(m_k = 3, x_k, e_k)$.

If we have that $\exists \lambda \in \{1, \ldots, n\} :\ \xi_\lambda \in \{2, 3\}$, then event $e_b := r$ is broadcast to the local supervisors $\mathcal{L}_i$ and the n-string π is forwarded to the Ω operator to synthesize the environmental failure mode ${}^{\epsilon_0}\mathcal{F}_\mathcal{S}$ for the SPECTER task planner as in eq. 5.4. The environment model ${}^{\epsilon_0}\mathcal{S}$ incorporates this failure by subtracting the failure automaton ${}^{\epsilon_0}\mathcal{F}_\mathcal{S}$ from the existing environment model such that ${}^{\epsilon_0}\mathcal{S}' := {}^{\epsilon_0}\mathcal{S} \setminus_{\asymp} {}^{\epsilon_0}\mathcal{F}_\mathcal{S}$. This operation, with quadratic complexity over the environment states, is much faster than the SPECTER prepossessing step (cubic complexity), enabling on-the-fly incorporation, since the operations performed on the environment matrix are sequential and independent. Thus, partitioning of the environment state space provides for parallel processing of these operations. Then, if a solution exists, the SPECTER task planner computes an optimal reconfigured task plan $\mathcal{T}_f$, that is guaranteed to satisfy the task specification γ, with starting state the current state of the environment. SPECTER produces optimal task plans based on the provided cost function. In the current, the optimality is with respect to the temporal cost $g_\mathcal{S}$. If no solution exists, an empty task plan is output, signifying an irrecoverable failure of the system. The process repeats itself until either the task specification has been satisfied or the failure renders the task specification infeasible.

5.4 Case Study

An illustration of this approach is found in the following example considering the Case study I presented in Section 2.5.1 and its supervisory control architecture depicted in Fig. 5.5. Each agent is assigned with a relevant TAHA or TADA based on its dynamics. In this case, the robot agent R_1 is controlled by its local supervisor $\mathcal{L}_1$ assigned with the TAHA $\mathcal{Y}_1$, the robot agent R_2 is controlled by its local supervisor

$\mathcal{L}_2$ assigned with the TAHA $\mathcal{Y}_2$, the Human Machine Interface (HMI) (considered as human agent W_1 in Section 2.5.1) is controlled by its local supervisor $\mathcal{L}_3$ assigned with the TADA $\mathcal{N}_\ni$ and the item is controlled by its local $\mathcal{L}_4$ assigned with the TADA $\mathcal{N}_4$. Assuming that there is a fault diagnoser observing the execution status and providing exogenous events to trigger transitions on TAHAs and TADAs. All local supervisors are observing the discrete state of the relevant TAHAs or TADAs to identify if there is a failure detected or if the execution is completed normally so as to inform the global supervisor about the execution status of the task plan.

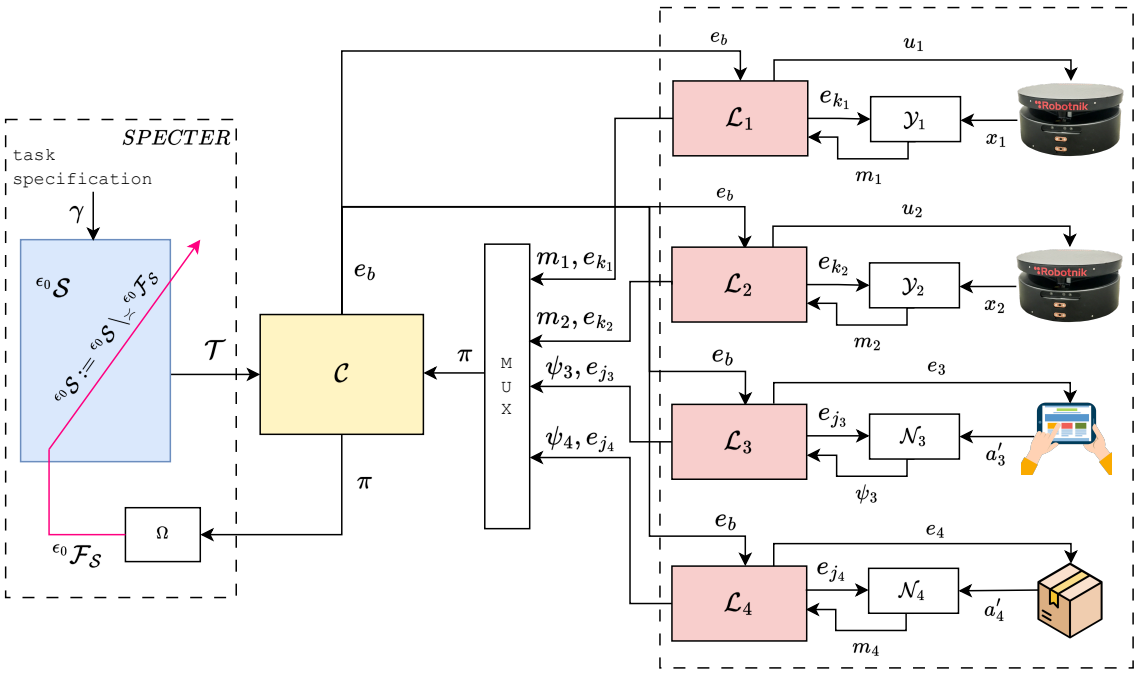


Figure 5.5: Supervisory control architecture of case study I.

Having defined the task specification, such that $\gamma = (B, b_1)$, and having the SPECTER task planner that provides the optimal solution that satisfies the task specification in a form of a module chain as shown in Fig. 2.20, the task plan $\mathcal{T}$ is fed to the global supervisor $\mathcal{C}$. The global supervisor $\mathcal{G}$ decomposes the sequence of modules to six individual modules. The global supervisor starts its process with the first module of the module chain and extracts the event e_1 of the module T_1. Then, the global supervisor starts broadcasting the event e_1 to all local supervisors $\mathcal{L}_1, \mathcal{L}_2, \mathcal{L}_3$ and $\mathcal{L}_4$.

At the moment, all agents are idle and the TAHAs and TADAs are on their starting state. All local supervisors receive the event e_1. The local supervisor $\mathcal{L}_2$ accepts the event e_1 and transmits e_1 to $\mathcal{Y}_2$, the TAHA of R_2, to initiate the normal execution of R_2. R_2 is idle meaning that $\mathcal{Y}_2$ is on its initial state, at state "0". A transition from state "0" to state "1" on $\mathcal{Y}_2$ has occurred, when $\mathcal{L}_2$ transmits e_1 to $\mathcal{Y}_2$. Then, the local supervisor $\mathcal{L}_2$ is informed that $\mathcal{Y}_2$ initiates its normal execution and provides u_2 to R_2. The clock of $\mathcal{Y}_2$ starts, while R_2 starts its navigation from docking station E to pick-up location A. The assigned time for R_2 to navigate from E to A must be 5 seconds.

Considering that an unmodeled failure has occurred on R_2 (shown in Fig. 2.21) that renders R_2 unable to reach the destination configuration of A. Then, the clock of $\mathcal{Y}_2$ exceeds the assigned time of 5 seconds and a transition from state "1" to state "2" is forced while the agent R_2 stops and the clock of $\mathcal{Y}_2$ stops.

This individual failure of agent R_1 indirectly affects the environmental capabilities due to rendering the inter-agent capabilities of Fig. 2.19a and Fig. 2.19c infeasible. In the meantime, $\mathcal{L}_2$, observing the discrete state of $\mathcal{Y}_2$, detects that there is an unmodeled failure detected on the agent R_2.

The multiplexer collects the discrete states of TAHAs and TADAs from their local supervisors and the corresponding events. In this case, $\mathcal{Y}_1$, $\mathcal{N}_3$ and $\mathcal{N}_4$ have not accepted the event e_1 and they are on their starting state, such that they are on the state "0" and no event is transmitted. However, $\mathcal{L}_2$ is on the unmodeled failure state, the state "2" and transmits the event e_1. Based on the input that the multiplexer collects from the local supervisors, it creates the vector π such that $\pi = (\{0, \emptyset\}, \{2, e_1\}, \{0, \emptyset\}, \{0, \emptyset\})$.

The vector π is fed to $\mathcal{C}$ so that the global supervisor is informed that the agent R_2 failed to complete the assigned task. Then, $\mathcal{C}$ transmits π to the Ω operator to synthesize the environmental failure mode automaton ${}^{\epsilon_0}\mathcal{F}_{\mathcal{S}}$ as defined in Definition 16. In this case, $x_\alpha = x_{0,\mathcal{S}} = DECA$, $e = e_1$ and $x_d = DACA$, hence $X_{\mathcal{F}_{\mathcal{S}}} = \{x_\beta \mid proj(x_\beta, b_1) \in \{proj(x_\alpha, b_1) = E,\ proj(x_d, b_1) = A\}\}$.

The SPECTER task planner receives the automaton ${}^{\epsilon_0}\mathcal{F}_{\mathcal{S}}$ from Ω operator and constructs the new environment model ${}^{\epsilon_0}\mathcal{S}'$ by subtracting ${}^{\epsilon_0}\mathcal{F}_{\mathcal{S}}$ from the existing ${}^{\epsilon_0}\mathcal{S}$. The updated $\mathcal{S}'$ disables the transitions of environment states that utilize the agent R_2. The SPECTER task planner calculates the new task plan that satisfies γ from the initial state $x_{0,\mathcal{S}} = DECA$. Then, the closed module chain $\overset{\circ}{\mathcal{T}}_f$ is reconfigured as shown in Fig. 2.22 utilizing agent R_1 and the modified task plan $\mathcal{T}_f$ is fed to $\mathcal{C}$ to initiate the task plan execution starting from module T_7. The process is repeated until γ is satisfied.

Now let us consider the case where the failure of R_2 (shown in Fig. 2.21) is a modeled failure, i.e. the agent is turned off due to battery failure meaning that R_2 is inactive and unable to execute any task at all. This renders R_2 unable to reach any particular destination. In the case of $\mathcal{Y}_2$ is on the execution state "1" and the event θ is provided by the fault diagnoser, then $\mathcal{Y}_2$ transitions from state "1" to modeled failure state "3". The agent R_2 is stopped and the clock of $\mathcal{Y}_2$ stops. In the meantime, $\mathcal{L}_2$, observing the discrete state of $\mathcal{Y}_2$, detects that there is a modeled failure detected on the agent R_2. In this case, $\mathcal{L}_2$ transmits the modeled failure state "3" of $\mathcal{Y}_2$ and the event θ to the multiplexer, such that the vector π is stated as $\pi = (\{0, \emptyset\}, \{3, \theta\}, \{0, \emptyset\}, \{0, \emptyset\})$.

The vector π is fed to $\mathcal{C}$ which is informed that agent R_2 has failed to execute the task and transmits the vector π to the Ω operator to synthesize the automaton ${}^{\epsilon_0}\mathcal{F}_{\mathcal{S}}$ (as in Definition 16). In this case, $x_\alpha = x_{0,\mathcal{S}} = DECA$, $e = \theta$ and $x_d = DACA$, hence $X_{\mathcal{F}_{\mathcal{S}}} = \{x_\beta \mid proj(x_\beta, b_1) \in \{proj(x_\alpha, b_1) = E,\ proj(x_d, b_1) = A\}\}$.

Subtracting ${}^{\epsilon_0}\mathcal{F}_{\mathcal{S}}$ from the existing ${}^{\epsilon_0}\mathcal{S}$, the new environment model ${}^{\epsilon_0}\mathcal{S}'$ is constructed. The updated $\mathcal{S}'$ disables the transitions of environment states that utilize the agent R_2. Then, the SPECTER task planner calculates the new task plan that satisfies γ from the initial state $x_{0,\mathcal{S}} = DECA$. Then, the closed module chain $\overset{\circ}{\mathcal{T}}_f$ is reconfigured as shown in Fig. 2.22 utilizing agent R_1 and the modified task plan $\mathcal{T}_f$ is fed to $\mathcal{C}$ to initiate the task plan execution starting from module T_7. The process is repeated until γ is satisfied.

Part IV

Conclusions and Future Research

6 Conclusions and Future Directions

6.1 Conclusions

This dissertation introduces a novel framework for task planning and control synthesis with reactive failure mode reconfiguration of possibly heterogeneous multi-agent systems. The conclusions of the thesis are presented below.

- We introduced a novel methodology and framework for automatic task planning for (possibly) heterogeneous multi-agent systems, called the SPECTER task planner and developed the corresponding software implementation. We proposed a formal framework utilizing the concept of NFAs with ϵ-transitions, to build the appropriate discrete abstractions to model the capabilities, constraints and failure modes of agents involved in the environment, as well as the inter-agent capabilities and constraints that emerge when multiple agents are present. In particular, given the capabilities, constraints and failure modes of the agents under the framework of NFAs with ϵ-transitions, SPECTER produces optimal solutions, providing the sequence of tasks for transporting the state of the environment from any initial state to a destination state satisfying the task specification. We have shown that the developed algorithms provide solutions (if such exists) with completeness and optimality guarantees. The results in Chapter 2 are supported through case studies that demonstrate the applicability, the effectiveness and the validity of the proposed methodology in successfully generating optimal executions for multi-agent systems with a predetermined set of individual capabilities and constraints as well as agent coupling capabilities and restrictions.
- We have investigated the potential of the significant reduction in the computational requirements. We have shown that a drastic reduction is observed by relaxing the completeness property requirement through the use of the efficient heuristics that provide sub-optimal solutions. In Chapter 2, the analysis of the proposed heuristic approach as well as the results of the case studies demonstrate that the proposed heuristic fully recovers the optimality.
- The formal framework for modeling the discrete abstractions of multi-agent systems has been implemented in manufacturing logistics. In particular, the workflow abstractions for manufacturing logistics optimization have been presented for a food service company and for a shoe making industry. The discrete abstractions that have been developed model the key features of the workflows and the resulting model abstraction was implemented on the SPECTER task planner. Several case studies were investigated with different numbers of robots engaged in the production line, buffer zones with various capacities, human workers involved in each workflow process grouped in teams or not; as well as many material and products utilized, transported and produced during the manufacturing procedure. Moreover, different abstractions of the same workflow have been developed. The results are supported through the case studies presented in Chapter 3 and Chapter 4.
- We introduced the concept of deterministic (TADA) and hybrid (TAHA) automata with time abstractions to, on the one hand provide a connection between time abstracting controllers and, on the other hand to provide a framework to model, analyze and control agents considering their modeled and unmodeled failure modes.

- We have introduced a new framework for multi-agent control synthesis with reactive failure-mode reconfiguration under a supervisory control framework. A heterogeneous team of agents comprising agents with purely discrete dynamics and agents with hybrid dynamics has been considered. The proposed framework leverages and expands the SPECTER task planner of Chapter 2 providing the capability for on-the-fly optimal task reconfiguration in the presence of failures, while ensuring the fulfillment of the task specification.
- Combining the SPECTER task planner with supervisory control theory, enables the reactive execution in dynamic environments and the concurrent execution of modules (e.g. in a multi-port setting), whereas improving the average computational complexity of both the pre-processing and planning phases. The additional effort in casting the problem as a module composition one, enables us to seamlessly use the generated module chain as a model system for building the appropriate supervisory controllers.

6.2 Future Directions

The proposed framework of automatic task planning for multi-agent systems can be expanded and enhanced adding more capabilities and features in the current approach in order to address diverse kinds of problems.

Some of the concepts that are considered as further research topics are presented below.

- In our study for multi-agent task planning, we have implemented only single port modules such that transitions can be performed by one agent at a time. We can expand this approach by integrating parallel execution utilizing multi-port modules for concurrent tasks to enable multiple agents to perform concurrent transitions. In the framework presented in Chapter 2, we observed that there are several actions that could be performed simultaneously from independent agents and this could reduce the execution time in order to fulfill the task. The main contribution of the future work will be that given a SPECTER task plan in the sense of a solution in a form of single-port module chain, to synthesize the multi-port module chain in an optimal way using all modules of SPECTER task plan fulfilling the given task specification. We will focus on developing an architecture that will take as input the solution (optimal/sub-optimal) provided by the SPECTER task planner to synthesize a parallelized task plan with synchronization on the tasks of the agents that are acting independently. By doing this, the temporal cost of the plan will decrease, and the agents are going to be utilized more efficiently. The goal is for the synchronization to be made on the inter-agent tasks; to identify which are the inter-agent tasks of the task plan; and formally express the concept though the module composition theory. We will investigate exploiting the module composition theory [85] to cast the problem of parallel execution of motion tasks as a module composition problem using multi-port modules. This is an interesting topic to be considered as a future direction.
- In the area of the discrete abstractions for high-level task planning, we will focus on expanding the addressable classes of problems that the framework of Chapter 2 can tackle to address more complicated problems and to be able to provide real-time solutions. Additionally, applications with different agents and layout arrangements will be considered as further investigations that will help to improve the proposed methodology, to identify outstanding issues as well as add more features

and capabilities so that the SPECTER task planner can be further utilized in various industrial cases.

- Future work also includes the formal specification and the performance analysis of the introduced supervisory controllers. Since the utilized low level time-abstracting controllers provide analytically guaranteed performance, our emphasis will be on determining correct-by-construction solutions to propagate the performance from the low level controllers to the high level supervisors.
- Another interesting topic for future investigation will be to seek ways to handle and drastically reduce the computational requirements of algorithms in Chapter 2 to calculate a solution, by exploiting the structure of the environment model. A major challenge to consider in the future is the state space explosion when modeling very complex environments. More specifically, the environment model takes even weeks or months to be constructed and it is impossible to run in real-time implementations. Studies that are working towards improving on this, seeking ways to decrease the complexity of the environment model using different techniques, such as sample-based searching. Sampling-based planners can quickly generate approximate solutions, however, they lose the completeness and optimality properties. On the other hand, search-based planners may require more time to find a solution – but this solution is optimal. To this end, sampling-based and search-based planners each have their strengths and weaknesses, which make them suitable for different types of task planning. To get the optimal solution is difficult, due to the "curse of dimensionality", implying increased computational complexity. To get a solution (if one exists) which is not required to be the optimal, sampling-based planners can quickly generate approximate solutions. Both options are interesting to be explored so as to provide additional features to our method.

BIBLIOGRAPHY

[1] S. M. LaValle, *Planning algorithms*. Cambridge University Press, 2006.

[2] E. Karpas and D. Magazzeni, "Automated planning for robotics," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 3, no. 1, pp. 417–439, 2020.

[3] J.-C. Latombe, *Robot motion planning*. Springer Science & Business Media, 2012, vol. 124.

[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.

[5] E. W. Dijkstra *et al.*, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[6] A. Stentz, "Optimal and efficient path planning for partially-known environments," in *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*. IEEE, 1994, pp. 3310–3317.

[7] ——, "The focussed D* algorithm for real-time replanning," in *International Joint Conference on Artificial Intelligence*, vol. 95, 1995, pp. 1652–1659.

[8] S. Koenig and M. Likhachev, "D*lite," in *Eighteenth National Conference on Artificial Intelligence*. USA: American Association for Artificial Intelligence, 2002, p. 476–483.

[9] R. Hill and S. Lafortune, "Scaling the formal synthesis of supervisory control software for multiple robot systems," in *2017 American Control Conference (ACC)*, 2017, pp. 3840–3847.

[10] J. Goryca and R. C. Hill, "Formal synthesis of supervisory control software for multiple robot systems," in *IEEE American Control Conference*. IEEE, 2013, pp. 125–131.

[11] G. A. Cardona and C.-I. Vasile, "Planning for heterogeneous teams of robots with temporal logic, capability, and resource constraints," *International Journal of Robotics Research*, vol. 43, no. 13, pp. 2089–2111, 2024.

[12] S. Hustiu, C. Mahulea, M. Kloetzer, and J.-J. Lesage, "On multi-robot path planning based on Petri net models and LTL specifications," *IEEE Transactions on Automatic Control*, pp. 1–8, 2024.

[13] Y. Kantaros and M. M. Zavlanos, "Sampling-based optimal control synthesis for multirobot systems under global temporal tasks," *IEEE Transactions on Automatic Control*, vol. 64, no. 5, pp. 1916–1931, 2018.

[14] ——, "Stylus*: A temporal logic optimal control synthesis algorithm for large-scale multi-robot systems," *The International Journal of Robotics Research*, pp. 812–836, 39(7), 2020.

[15] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas, "Where's waldo? Sensor-based temporal logic motion planning," in *IEEE International Conference on Robotics and Automation*, 2007, pp. 3116–3121.

[16] ——, "Temporal-logic-based reactive mission and motion planning," *IEEE Transactions on Robotics*, vol. 25, no. 6, pp. 1370–1381, 2009.

[17] M. Guo and D. V. Dimarogonas, "Task and motion coordination for heterogeneous multiagent systems with loosely coupled local tasks," *IEEE Transactions on Automation Science and Engineering*, vol. 14, no. 2, pp. 797–808, 2016.

[18] S. L. Smith, J. Tŭmová, 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.

[19] I. Filippidis, D. V. Dimarogonas, and K. J. Kyriakopoulos, "Decentralized multi-agent control from local LTL specifications," in *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*. IEEE, 2012, pp. 6235–6240.

[20] A. I. Rikos, T. Charalambous, and C. N. Hadjicostis, "Distributed weight balancing over digraphs," *IEEE Transactions on Control of Network Systems*, vol. 1, no. 2, pp. 190–201, 2014.

[21] S. Sundaram and C. N. Hadjicostis, "Control and estimation in finite state multi-agent systems: A finite field approach," *IEEE Transactions on Automatic Control*, vol. 58, pp. 60–73, 2013.

[22] N. Xu, T. Peng, D. Liu, and J. Li, "Temporal logic control synthesis for distributed multi-agent cooperative tasking," *Journal of Physics: Conference Series*, vol. 2216, no. 1, p. 012061, Mar. 2022. [Online]. Available: https://dx.doi.org/10.1088/1742-6596/2216/1/012061

[23] Y. Chen, X. C. Ding, A. Stefanescu, and C. Belta, "A formal approach to deployment of robotic teams in an urban-like environment," in *10th International Symposium on Distributed Autonomous Robotic Systems*. Springer, 2013, pp. 313–327.

[24] M. Kloetzer and C. Belta, "Automatic deployment of distributed teams of robots from temporal logic motion specifications," *IEEE Transactions on Robotics*, vol. 26, no. 1, pp. 48–61, 2009.

[25] M. M. Quottrup, T. Bak, and R. Zamanabadi, "Multi-robot planning: A timed automata approach," in *IEEE International Conference on Robotics and Automation*, vol. 5, 2004, pp. 4417–4422.

[26] M. Karimadini and H. Lin, "Guaranteed global performance through local coordinations," *Automatica*, vol. 47, no. 5, pp. 890–898, 2011.

[27] K. He, M. Lahijanian, L. E. Kavraki, and M. Y. Vardi, "Reactive synthesis for finite tasks under resource constraints," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 5326–5332.

[28] H. Lin, "Mission accomplished: An introduction to formal methods in mobile robot motion planning and control," *Unmanned Systems*, vol. 2, no. 02, pp. 201–216, 2014.

[29] J. Tůmová and D. V. Dimarogonas, "A receding horizon approach to multi-agent planning from local LTL specifications," in *IEEE American Control Conference*, 2014, pp. 1775–1780.

[30] P. Schillinger, M. Bürger, and D. V. Dimarogonas, "Simultaneous task allocation and planning for temporal logic goals in heterogeneous multi-robot systems," *The International Journal of Robotics Research*, vol. 37, no. 7, pp. 818–838, 2018.

[31] M. Kloetzer and C. Mahulea, "Multi-robot path planning for syntactically co-safe LTL specifications," in *13th International Workshop on Discrete Event Systems (WODES)*. IEEE, 2016, pp. 452–458.

[32] S. G. Loizou and K. J. Kyriakopoulos, "Automatic synthesis of multi-agent motion tasks based on LTL specifications," in *2004 43rd IEEE Conference on Decision and Control (CDC)*, vol. 1. IEEE, 2004, pp. 153–158.

[33] X. Luo and M. M. Zavlanos, "Temporal logic task allocation in heterogeneous multirobot systems," *IEEE Transactions on Robotics*, vol. 38, no. 6, pp. 3602–3621, 2022.

[34] K. Leahy, Z. Serlin, C.-I. Vasile, A. Schoer, A. M. Jones, R. Tron, and C. Belta, "Scalable and robust algorithms for task-based coordination from high-level specifications (ScRATCHeS)," *IEEE Transactions on Robotics*, vol. 38, no. 4, pp. 2516–2535, 2021.

[35] Z. Chen and Z. Kan, "Real-time reactive task allocation and planning of large heterogeneous multi-robot systems with temporal logic specifications," *The International Journal of Robotics Research*, 2024.

[36] S. Lafortune, "Discrete event systems: Modeling, observation, and control," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 2, no. 1, pp. 141–159, 2019.

[37] R. Alur, C. Courcoubetis, T. A. Henzinger, and P.-H. Ho, "Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems," in *International Hybrid Systems Workshop*. Springer, 1993, pp. 209–229.

[38] R. Alur and D. L. Dill, "A theory of timed automata," *Theoretical computer science*, vol. 126, no. 2, pp. 183–235, 1994.

[39] A. Puri and P. Varaiya, "Verification of hybrid systems using abstractions," in *Hybrid Systems II 2*. Springer, 1995, pp. 359–369.

[40] N. Lynch, R. Segala, and F. Vaandrager, "Hybrid I/O automata," *Information and computation*, vol. 185, no. 1, pp. 105–157, 2003.

[41] C. Galindo, J.-A. Fernandez-Madrigal, and J. Gonzalez, "Improving efficiency in mobile robot task planning through world abstraction," *IEEE Transactions on Robotics*, vol. 20, no. 4, pp. 677–690, 2004.

[42] C. Belta, V. Isler, and G. J. Pappas, "Discrete abstractions for robot motion planning and control in polygonal environments," *IEEE Transactions on Robotics*, vol. 21, no. 5, pp. 864–874, 2005.

[43] 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.

[44] N. T. Dantam, Z. K. Kingston, S. Chaudhuri, and L. E. Kavraki, "An incremental constraint-based framework for task and motion planning," *The International Journal of Robotics Research*, vol. 37, no. 10, pp. 1134–1151, 2018.

[45] J. A. DeCastro, V. Raman, and H. Kress-Gazit, "Dynamics-driven adaptive abstraction for reactive high-level mission and motion planning," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2015, pp. 369–376.

[46] J. McMahon and E. Plaku, "Sampling-based tree search with discrete abstractions for motion planning with dynamics and temporal logic," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2014, pp. 3726–3733.

[47] A. Curtis, T. Silver, J. B. Tenenbaum, T. Lozano-Pérez, and L. Kaelbling, "Discovering state and action abstractions for generalized task and motion planning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, no. 5, 2022, pp. 5377–5384.

[48] "Gurobi optimization," Available at URL: https://www.gurobi.com/, 2023, [Online]. Accessed: 30 November 2024.

[49] "Nvidia cuOpt," Available at URL: https://www.nvidia.com/en-us/ai-data-science/products/cuopt/, 2024, [Online]. Accessed: 30 November 2024.

[50] M. Aramon, G. Rosenberg, E. Valiante, T. Miyazawa, H. Tamura, and H. G. Katzgraber, "Physics-inspired optimization for quadratic unconstrained problems using a digital annealer," *Frontiers in Physics*, vol. 7, p. 48, 2019.

[51] S. Tanaka, Y. Matsuda, and N. Togawa, "Theory of ising machines and a common software platform for ising machines," in *25th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2020, pp. 659–666.

[52] "OpenJij," Available at URL: https://www.openjij.org/, 2023, [Online]. Accessed: 30 November 2024.

[53] C. Belta, B. Yordanov, and E. A. Gol, *Formal methods for discrete-time dynamical systems*. Springer, 2017, vol. 89.

[54] P. Tabuada, *Verification and control of hybrid systems: A symbolic approach*. Springer Science & Business Media, 2009.

[55] C. Baier and J.-P. Katoen, *Principles of model checking*. MIT press, 2008.

[56] J. Lygeros, C. Tomlin, and S. Sastry, "Controllers for reachability specifications for hybrid systems," *Automatica*, vol. 35, no. 3, pp. 349–370, 1999. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0005109898001939

[57] S. G. Loizou and E. D. Rimon, "Mobile robot navigation functions tuned by sensor readings in partially known environments," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 3803–3810, 2022.

[58] P. J. Ramadge and W. M. Wonham, "Supervisory control of a class of discrete event processes," *SIAM Journal on Control and Optimization*, vol. 25, no. 1, pp. 206–230, 1987.

[59] C. N. Hadjicostis, *Estimation and inference in discrete event systems*. Springer, 2020.

[60] H. Wong-Toi and G. Hoffmann, "The control of dense real-time discrete event systems," in *[1991] Proceedings of the 30th IEEE Conference on Decision and Control*, 1991, pp. 1527–1528 vol.2.

[61] O. Maler, A. Pnueli, and J. Sifakis, "On the synthesis of discrete controllers for timed systems," in *STACS 95*, E. W. Mayr and C. Puech, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1995, pp. 229–242.

[62] R. Kumar and M. A. Shayman, "Supervisory control of real-time systems using prioritized synchronization," in *Hybrid Systems III*, R. Alur, T. A. Henzinger, and E. D. Sontag, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 350–361.

[63] M. Spathopoulos, "On supervisory control for timed automata using urgency," in *Proceedings of the 9th IEEE International Conference on Methods and Models in Automation and Robotics*, 2003, pp. 863–869.

[64] F. Liu and H. Lin, "Reliable supervisory control for general architecture of decentralized discrete event systems," *Automatica*, vol. 46, no. 9, pp. 1510–1516, 2010.

[65] K. Rohloff and S. Lafortune, "On the synthesis of safe control policies in decentralized control of discrete-event systems," *IEEE Transactions on Automatic Control*, vol. 48, no. 6, pp. 1064–1068, 2003.

[66] R. Kumar and S. Takai, "Inference-based ambiguity management in decentralized decision-making: Decentralized control of discrete event systems," *IEEE Transactions on Automatic Control*, vol. 52, no. 10, pp. 1783–1794, 2007.

[67] F. Lin and W. M. Wonham, "On observability of discrete-event systems," *Information sciences*, vol. 44, no. 3, pp. 173–198, 1988.

[68] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. Teneketzis, "Diagnosability of discrete-event systems," *IEEE Transactions on Automatic Control*, vol. 40, no. 9, pp. 1555–1575, 1995.

[69] S. H. Zad, R. H. Kwong, and W. M. Wonham, "Fault diagnosis in discrete-event systems: Framework and model reduction," *IEEE Transactions on Automatic Control*, vol. 48, no. 7, pp. 1199–1212, 2003.

[70] C. Keroglou and C. N. Hadjicostis, "Distributed fault diagnosis in discrete event systems via set intersection refinements," *IEEE Transactions on Automatic Control*, vol. 63, no. 10, pp. 3601–3607, 2018.

[71] E. Athanasopoulou and C. Hadjicostis, "Decentralized failure diagnosis in discrete event systems," in *2006 American Control Conference*, 2006, pp. 6 pp.–.

[72] Y. Wu and C. N. Hadjicostis, "Algebraic approaches for fault identification in discrete-event systems," *IEEE Transactions on Automatic Control*, vol. 50, no. 12, pp. 2048–2055, 2005.

[73] S. Ware and R. Su, "Incremental scheduling of discrete event systems," in *2016 13th International Workshop on Discrete Event Systems (WODES)*, 2016, pp. 147–152.

[74] M. Schmidt and J. Lunze, "A framework for active fault-tolerant control of deterministic I/O automata," *IFAC Proceedings Volumes*, vol. 47, no. 2, pp. 446–452, 2014.

[75] M. Zgorzelski and J. Lunze, "Feedforward and state-feedback control of deterministic I/O automata," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 13 426–13 433, 2017.

[76] M. Schuh and J. Lunze, "Feedback control of nondeterministic input/output automata," in *53rd IEEE Conference on Decision and Control*, 2014, pp. 6737–6743.

[77] M. Guo, J. Tumova, and D. V. Dimarogonas, "Cooperative decentralized multi-agent control under local LTL tasks and connectivity constraints," in *53rd IEEE Conference on Decision and Control*, 2014, pp. 75–80.

[78] Y. Chen, X. C. Ding, and C. Belta, "Synthesis of distributed control and communication schemes from global LTL specifications," in *2011 50th IEEE Conference on Decision and Control and European Control Conference*, 2011, pp. 2718–2723.

[79] C. C. Constantinou and S. G. Loizou, "Automatic controller synthesis of motion-tasks with real-time objectives," in *2018 IEEE Conference on Decision and Control (CDC)*, 2018, pp. 403–408.

[80] S. Loizou and K. Kyriakopoulos, "Automated planning of motion tasks for multi-robot systems," in *Proceedings of the 44th IEEE Conference on Decision and Control*, 2005, pp. 78–83.

[81] K. Elimelech, L. E. Kavraki, and M. Y. Vardi, "Efficient task planning using abstract skills and dynamic road map matching," in *The International Symposium of Robotics Research*. Springer, 2022, pp. 487–503.

[82] D. McDermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, and D. Wilkins, "PDDL - The planning domain definition language," *Technical Report*, 1998.

[83] M. Fox and D. Long, "PDDL2.1: An extension to PDDL for expressing temporal planning domains," *Journal of Artificial Intelligence Research*, vol. 20, pp. 61–124, 2003.

[84] R. E. Fikes and N. J. Nilsson, "STRIPS: A new approach to the application of theorem proving to problem solving," *Artificial Intelligence*, vol. 2, no. 3-4, pp. 189–208, 1971.

[85] S. Tripakis, "Automated module composition," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2003, pp. 347–362.

[86] S. G. Loizou, "The navigation transformation," *IEEE Transactions on Robotics*, vol. 33, no. 6, pp. 1516–1523, 2017.

[87] "Business process optimization (BPO) module," https://github.com/rcdslabcut/mod.sw.bpo, accessed: November 2023.

[88] C. G. Cassandras and S. Lafortune, *Introduction to discrete event systems*. Springer, 2008.

[89] J. E. Hopcroft, R. Motwani, and J. D. Ullman, *Automata theory, languages, and computation*. Pearson, 2006.

[90] B. Khoussainov and A. Nerode, *Automata theory and its applications*. Springer Science & Business Media, 2012, vol. 21.

[91] L. W. Beineke and J. S. Bagga, *Line graphs and line digraphs*. Springer, 2021.

[92] M. Barbehenn, "A note on the complexity of Dijkstra's algorithm for graphs with weighted vertices," *IEEE Transactions on Computers*, vol. 47, no. 2, p. 263, 1998.

[93] M. Droste, W. Kuich, and H. Vogler, *Handbook of weighted automata*. Springer Science & Business Media, 2009.

[94] M. Lahijanian, S. B. Andersson, and C. Belta, "Formal verification and synthesis for discrete-time stochastic systems," *IEEE Transactions on Automatic Control*, vol. 60, no. 8, pp. 2031–2045, 2015.

[95] A. A. Tziola and S. G. Loizou, "Autonomous task planning for heterogeneous multi-agent systems," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, 2023, pp. 3490–3496.

APPENDICES

APPENDIX I

Template for encoding in SPECTER

```
{
  "Environment": {
    "Areas of interest": {
      "Name": [ "Area1", "Area2", ... ],
      "Number": [ 1, 2, ... ]
    },
    "Agents": {
      "agent_1": [ <state_space_of_agent_1> ],
      "agent_2": [ <state_space_of_agent_2> ],
      . . .
      "agent_n": [ <state_space_of_agent_n> ]
    }
  },

  "Capabilities": {
    "agent_1": {
      "stateA": { "stateB":cost, "stateC":cost },
      "stateB": { "stateC":cost, "stateD":cost, stateE:cost },
      "stateC: { ... },
      "stateD: { ... },
      "stateE": { ... }
    },
    "agent_2": {
      "stateA": { "stateB":cost, "stateC":cost },
      "stateB": { "stateC":cost, "stateD":cost, stateE:cost },
      . . .
    },
    . . .
    "agent_n": {
      . . .
    },

  "Constraints": {
    "agent_1": {
      "stateA": { "stateB", "stateC" },
      "stateB": { "stateC", "stateD", stateE },
```

```
        "stateC: { ... },
        "stateD: { ... },
        "stateE": { ... }
      },
      "agent_2": {
        "stateA": { "stateB", "stateC" },
        "stateB": { "stateC", "stateD", stateE },
        . . .
      },
      . . .
      "agent_n": {
        . . .
      },
    },
    "Inter-Agent Capabilities": {
      "1": {
        "cost":[ <number> ],
        "agent_1": [ state, state ],
        "agent_2": [ state, state ],
        "agent_3": [ state, state ]
      },
      "2": {
        "cost":[ <number> ],
        "agent_2": [ state, state ],
        "agent_3": [ state, state ]
      },
      . . .
    },
    "Inter-Agent Constraints": {
      "1": {
        "agent_1": [ state, state ],
        "agent_2": [ state, state ],
        "agent_3": [ state, state ]
      },
      "2": {
        "agent_1": [ state, state ],
        "agent_2": [ state, state ],
        "agent_3": [ state, state ]
      },
      . . .
  },
```

```
      "Current positions": {
          "agent_1": [ starting_state ],
          "agent_2": [ starting_state ],
          . . .
          "agent_n": [ starting_state ]
      },
      "Goal positions": {
          "agent_1": [ goal_state ],
          "agent_2": [ goal_state ],
          . . .
          "agent_n": [ goal_state ]
      }
    },
      "Failure":{
        "0": {
        "agent_1": [ state, state ]
        }
    }
}
```