Autonomous Task Planning for Heterogeneous Multi-Agent Systems*

Anatoli A. Tziola and Savvas G. Loizou

Abstract—This paper presents a solution to the automatic task planning problem for multi-agent systems. A formal framework is developed based on Nondeterministic Finite Automata with $\epsilon$-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. A case study is provided to demonstrate the effectiveness and validity of the methodology.

I. INTRODUCTION

Multi-agent systems operating autonomously in dynamical environments to perform complicated tasks have been one of the major areas of research interest during the last decade. 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. Some of the most common approaches include Linear Temporal Logic (LTL), sampling-based approaches and domain definition languages. Many of the existing works use LTL formulas to develop bottom-up approaches [1]–[4] where local LTL expressions are assigned to robots, or top-down approaches [5], [6], [7], where a global task is decomposed into independent subtasks that are treated separately by each agent.

In [1], a high-level plan is found by a discrete planner which seeks the set of system transitions that ensure the satisfaction of a logic formula. In [8], a formal method based on LTL has been developed to model specifications and implement centralized planning for multi-agent systems. Some related works suggest that hierarchical abstraction techniques for single-agent systems can be extended to multi-agent systems using parallel compositions [6], [7]. The fact that the transitions should be on the common events to allow parallel execution is very restrictive. In [3], temporal logic formulas are utilized to specify sub-formulas that could be executed in parallel by the agents, without a global task definition. In [9], the authors tackle the multi-robot task allocation problem under constraints defined by LTL formalism to concurrently plan tasks for robot agents. In [10], a formal method based on LTL has been deployed to model multi robot motion planning specifications. Common among the above works is the adaptation of formal verification techniques for motion planning and controller synthesis. The two main limitations of the above works are the exponential growth of the state space even for relatively simple problems and the extra computations required to express LTL formulas in Büchi automata. In [11], authors propose a formal synthesis of supervisory control software for multi-robot systems, while the scalability of the approach was improved in [12]. On the other hand, implementing Planning Domain Definition Language (PDDL) [13], [14] would be time consuming for real time applications.

In [15], a sampling-based approach is presented using directed trees to approximate the state space and transitions of synchronous product automata. The sampling process is guided by transitions that belong to the shortest path to the accepting states. However, these algorithms provide no solution quality guarantee. Our proposed approach seeks the optimal shortest path in a weighted graph and a case study focusing on the logistics flow aspects was chosen.

The main contribution of this work are:

- System modeling using nondeterministic finite automata with $\epsilon$-transitions [16] expressing the system’s behavior and combining the agents capabilities and constraints at the individual and group level, 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 cycle.
- Determination of reduced complexity sub-optimal solutions to the task planning problem.
- Incorporation of failure modes on-the-fly, after building the global system model (i.e. without the need to repeat the costly pre-processing step).

This work 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. This work builds on top and expands the results of [17]. The problem is posed as a special case of module composition problem (MCP) [18] and then reduced to a combinatorial optimization problem.

*This work was partially supported by European Union’s H2020 research and innovation program under grant agreement 951813 (Better Factory).

The authors are with the Department of Mechanical Engineering and Materials Science and Engineering, Cyprus University of Technology, Limassol, CYPRUS. {anatoli.tziola, savvas.loizou}@cut.ac.cy.

979-8-3503-2356-8/23/$31.00 ©2023 IEEE 3490
of shortest dipath, which can be solved in polynomial time. The optimal module chain, combined with the Supervisory Control Theory (SCT) [19] can then be applied to synthesize control laws and communication strategies for efficiently accomplishing a global tasks [20]. Agents’ navigation can be handled by a navigation methodology with performance guarantees that ensure compositionality, like the navigation transformation [21]. Agents’ control and navigation are not under the scope of the current work. The active source code of the software developed is available at [22].

The rest of the paper is organized as follows: Section II presents the necessary preliminary notions, section III presents the problem formulation using the ε₀-NFA formalism while section IV exploits the ε₀-NFA formalism to reduce the problem to a Module Composition one. Section V presents the algorithms and the analysis of the methodology while section VI presents a case study to demonstrate the operation of SPECTER. Section VII concludes the paper.

II. PRELIMINARIES

A. 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 ∪ B and A ∩ B whereas set subtraction of set B from set A is denoted as A \ B. Operator ∧ denotes conjunction.

We will use the definition of Deterministic Finite Automata (DFA) by [16] and represent them as G = (X, E, f, Γ, x₀, G, X, m, G).

Definition 1 (ε₀-NFA). An ε₀-NFA is a Nondeterministic Finite Automaton with ε-transitions only from the initial state, defined as a six-tuple, ε₀G = (XG ∪ {x₀}: EG ∪ {ε}, ε₀fG, ΓG, xo,G, XM,G), where ε₀fG : XG ∪ {x₀} × EG ∪ {ε} → 2XG such that ∀(x, e) ∈ XG × EG : fG(x, e) = ε₀fG(x, e), where the variables with “G” subscript are as in the DFA Definition and x₀ ∉ XG is the initial state.

Corollary 1. If ε₀G is an ε₀-NFA then it can be converted to the DFA G by removing x₀ along with the associated ε transitions and assigning an x₀,G ∈ XG as an initial state.

Proof. Please see the Proof of the same in [23]

Based on Corollary 1, we construct the following:

Definition 2. Let ε₀G be an ε₀-NFA and x₀,G ∈ XG a state in XG that we want to assign as an initial state. Then Δ(ε₀G, x₀,G) ≔ G, where G is a DFA. Operator Δ performs the conversion described in Corollary 1.

We need to associate a cost function with the transitions:

Definition 3 (Transition Cost Function). A transition cost function for an event set EG, is defined as yG : EG → R⁺.

We define the inverse transition function as:

Definition 4 (Inverse Transition Function). Let G be a DFA. Define f⁻¹ G : XG × EG → XG to be the inverse transition function, if such exists, such that fG(xG, e) = yG ∧ f⁻¹ G(yG, e) = xG for some xG : e ∈ G(xG).

For the automata operations that will be required in the sequel, we need to introduce the following concept:

Definition 5 (Compatible ε₀-NFAs). Let ε₀G and ε₀B be ε₀-NFAs. Let EC = EG ∩ EB. ε₀G and ε₀B are compatible if ∀e ∈ EC then, fG(xG, e) = fB(xB, e) and f⁻¹ G(yG, e) = f⁻¹ B(yB, e) for some xG : e ∈ G(xG), xB : e ∈ B(xB), yG : e ∈ G(yG), and yB : e ∈ B(yB). In such case it will also be true that xG = xB and yG = yB.

We denote such a compatibility relation as ε₀G ≍ ε₀B.

Note that the above definition effectively forces the endpoints of common events to be common states.

B. Operations on compatible automata

Here we introduce a custom set of the basic operations on compatible ε₀-NFAs: union, subtraction and concatenation. The introduced operations differ from the ones found in the automata literature [16], [24], [25] and provide the necessary functionality for the subsequent developments.

Definition 6 (Union of Compatible Automata). For the ε₀-NFAs ε₀G and ε₀B, assume ε₀G ≍ ε₀B. Define the compatible automata union ε₀P ≌ ε₀G ∪ ε₀B to be the six-tuple ε₀P = (Xp ∪ {x₀}, Ep ∪ {ε}, ε₀fP, Θ, x₀, Xp, m, p), where Xp = XG ∪ XB, Ep = EG ∪ EB, ε₀fP = : Xp ∪ {x₀} × Ep ∪ {ε} → 2Xp, such that ∀(x, e) ∈ Xp × Ep : ε₀fP(x, e) = fP(x, e) and ∀(x, e) ∈ Xp × Ep : ε₀fP(x, e) = fP(x, e), Θ : Xp → 2Ep, x₀ ∉ Xp and Xp, m, p = XM.G ∪ XM.B.

In contrast with the standard union operation on automata presented in the literature, the above operation produces the union instead of the cartesian product state space.

Definition 7 (Subtraction of Compatible Automata). For the ε₀-NFAs ε₀G and ε₀B, assume ε₀G ≍ ε₀B. Define the compatible automata subtraction ε₀C ≌ ε₀G \ ε₀B to be the six-tuple ε₀C = (Xc ∪ {x₀}, Ec ∪ {ε}, ε₀fC, Γ, x₀, Xc, m, c), where Xc = XG, Ec = EG, ε₀fC = : Xc ∪ {x₀} × Ec ∪ {ε} → 2Xc, s.t. ∀(x, e) ∈ Xc × Ec : ε₀fC(x, e) = fC(x, e), Γ : Xc → 2Ec, x₀ ∉ Xc and Xc, m, c = XM.G \ XM.B.

Note that this is more in line with the set difference operator than with the intersection with the language complement that is used in regular languages.

Definition 8 (Concatenation of Compatible Automata). For the ε₀-NFAs ε₀G and ε₀B, assume ε₀G ≍ ε₀B. Define the compatible automata concatenation ε₀Φ ≌ ε₀G II ε₀B to be the six-tuple ε₀Φ = (XΦ ∪ {x₀}, EΦ ∪ {ε}, ε₀fΦ, ΓΦ, x₀, XΦ, m, Φ), where XΦ = {uv | u ∈ XG, v ∈ XB}, EΦ = EG ∪ EB, ε₀fΦ = : XΦ ∪ {x₀} × EΦ ∪ {ε} → 2XΦ, s.t. ∀(x, e) ∈ XΦ × EΦ : ε₀fΦ(x, e) = fΦ(x, e), ΓΦ : XΦ → 2EΦ, x₀ ∉ XΦ, XΦ, m, Φ = {uv | u ∈ XM.G \ XM.B \ XΦ}.

1Note that due to Corollary 1 we have that ε₀fC(xG, e) ≡ fC(xG, e) and ε₀fB(xB, e) ≡ fB(xB, e).
Note that the operation in Definition 8 above, is similar to the well known parallel composition operation (see e.g. [16]) 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).

The concept of compatible automata, gives rise to the following properties related to Definitions 6, 7 and 8:

**Corollary 2.** The automata resulting from the operations defined in Definitions 6, 7 and 8 are $\epsilon_0$-NFAs.

**Proof.** Please see the Proof of the same in [23].

We need to introduce the following operator that addresses individual states of concatenated states of Definition 8.

**Definition 9** (Projection Operator). Assume $x \in X_G$ is a state of $G$ with $|x| = n$ and let $x_i$ denote the $i$’th element of $x$. Define the projector $b$ as an $n$-bit binary and $b_i$ its $i$’th bit. The projection operator is defined as $proj(x, b) \triangleq \{x_i|b_i = 1\}$.

We will need some definitions from MCP literature as well as some new ones for our development. Using the module definition by [18], we state the module in an appropriate form for our development.

**Definition 10** (Single Port Module). Let $G$ be a DFA. Consider the finite set of ports $P = P_{in} \cup P_{out}$, where $P_{in}$ be a non-empty finite set of input ports and $P_{out}$ be 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 $T \triangleq \{p, e, q\}$, 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.

The cost associated with the module $T$ is defined as: $c(T) \triangleq y_G(e)$. We can also define the inverted module $T^{-1}$ where its input becomes its output and vice versa such that Definition 4 holds.

### III. PROBLEM FORMULATION

#### A. Agent Model

Let $n$ be the total number of agents and $\epsilon_0 A$ be the finite set of the agents $\epsilon_0$-NFAs, where $n = |\epsilon_0 A|$. We represent the $i$’th agent as $\epsilon_0 A_i$, $i \in \{1, \ldots, n\}$.

1) **Individual Agent Capabilities:** Let $\kappa$ be the total number of individual capabilities of $\epsilon_0 A_i$ and $\epsilon_0 M_{\beta,i}$ the $\beta$’th individual capability of $\epsilon_0 A_i$, where $\beta \in \{1, \ldots, \kappa\}$.

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 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 $\epsilon_0$-NFA $\epsilon_0 F_i$ such that $F_i = \{x, x'\}$ and $E_i = \{e\}$.

We model the agent’s capabilities as the subtraction of the agent failure from the union of individual agent capabilities.

3) **Agent Capabilities:** Considering $\kappa$ compatible $\epsilon_0$-NFAs such that $\epsilon_0 M_{\alpha,i} \preceq \epsilon_0 M_{\beta,i}$, $\alpha \neq \beta$ with $\alpha, \beta \in \{1, \ldots, \kappa\}$ and $\epsilon_0 F_i \preceq \epsilon_0 M_{\beta,i}$, the capabilities of $\epsilon_0 A_i$ are modeled by the $\epsilon_0$-NFA $\epsilon_0 K_i$ as:

$$\epsilon_0 K_i \triangleq \bigcup_{\beta=1}^{\kappa} \epsilon_0 M_{\beta,i} \setminus \epsilon_0 F_i.$$

4) **Individual Agent Constraints:** Let $\lambda$ be the total number of individual constraints of $\epsilon_0 A_i$ and $\epsilon_0 N_{\xi,i}$, the $\xi$’th individual constraint of $\epsilon_0 A_i$, where $\xi \in \{1, \ldots, \lambda\}$.

We model the agent’s constraints as the union of individual agent constraints.

5) **Agent Constraints:** Considering $\lambda$ compatible $\epsilon_0$-NFAs such that $\epsilon_0 N_{\xi,i} \preceq \epsilon_0 N_{\eta,i}$, $\xi \neq \eta$ with $\eta \in \{1, \ldots, \lambda\}$, the constraints of $\epsilon_0 A_i$ are modeled by the $\epsilon_0$-NFA $\epsilon_0 D_i$ as:

$$\epsilon_0 D_i \triangleq \bigcup_{\eta=1}^{\lambda} \epsilon_0 N_{\xi,i}.$$

Considering $\epsilon_0 K_i \preceq \epsilon_0 D_i$, the agent $i$’th agent is modeled by the $\epsilon_0$-NFA of $\epsilon_0 K_i$ after subtracting $\epsilon_0 D_i$. Thus, $\epsilon_0 A_i \triangleq \epsilon_0 K_i \setminus \epsilon_0 D_i$.

#### B. Environment Model

We model the environmental capabilities as the concatenation of agents’ capabilities to express the allowed environment state transitions.

1) **Environmental Capabilities:** For the $\epsilon_0$-NFAs $\epsilon_0 K_i$ and $\epsilon_0 K_j$, $i \neq j$ with $i, j \in \{1, \ldots, n\}$, let $\epsilon_0 K_i \preceq \epsilon_0 K_j$. The environmental capabilities are modeled by the $\epsilon_0$-NFA $\epsilon_0 \mathcal{K}$ as:

$$\epsilon_0 \mathcal{K} \triangleq \bigcup_{i=1}^{n} \epsilon_0 K_i.$$

We model the environmental constraints as the concatenation of agents’ constraints to express the not-permitted environment state transitions.

2) **Environmental Constraints:** For the $\epsilon_0$-NFAs $\epsilon_0 D_i$ and $\epsilon_0 D_j$, $i \neq j$ with $i, j \in \{1, \ldots, n\}$, let $\epsilon_0 D_i \preceq \epsilon_0 D_j$. The environmental constraints are modeled by the $\epsilon_0$-NFA $\epsilon_0 \mathcal{D}$ as:

$$\epsilon_0 \mathcal{D} \triangleq \bigcup_{i=1}^{n} \epsilon_0 D_i.$$

Then, inter-agents capabilities and inter-agents constraints are modeled as follows:

3) **Inter-Agents Capabilities:** Let $\mathcal{C} \subseteq \epsilon_0 \mathcal{A}$ with $||\mathcal{C}|| \leq n$ be a set of compatible agent $\epsilon_0$-NFAs. Then, the $\epsilon_0$-NFA $\epsilon_0 \mathcal{K}_A \subseteq \bigcup_{i \in \mathcal{C}} \epsilon_0 A_i$ denotes the inter-agents capabilities2 between the members of $\mathcal{C}$.

4) **Inter-Agents Constraints:** Let $\mathcal{C} \subseteq \epsilon_0 \mathcal{A}$ with $||\mathcal{C}|| \leq n$ be a set of compatible agent $\epsilon_0$-NFAs. Then, the $\epsilon_0$-NFA $\epsilon_0 \mathcal{D}_A \subseteq \bigcup_{i \in \mathcal{C}} \epsilon_0 A_i$ denotes the inter-agents constraints between the members of $\mathcal{C}$.

We model the global capabilities by adding the inter-agent capabilities to the environmental capabilities.

5) **Global Capabilities:** For the $\epsilon_0$-NFAs $\epsilon_0 \mathcal{K}$ and $\epsilon_0 \mathcal{K}_A$, let $\epsilon_0 \mathcal{K} \preceq \epsilon_0 \mathcal{K}_A$. The global capabilities are modeled by the $\epsilon_0$-NFA $\epsilon_0 \mathcal{K}$ defined as: $\epsilon_0 \mathcal{K} \triangleq \epsilon_0 \mathcal{K} \cup \epsilon_0 \mathcal{K}_A$.

We model the global constraints by adding the inter-agent constraints to the environmental constraints.

2Inter-Agents Failure Modes can be defined so as to restrict Inter-Agents Capabilities
6) **Global Constraints:** For the $\epsilon_0$-NFAs $\epsilon_0D$ and $\epsilon_0DA$, let $\epsilon_0D \equiv \epsilon_0DA$. The global constraints are modeled by the $\epsilon_0$-NFA $\epsilon_0D$ defined as: $\epsilon_0D \triangleq \epsilon_0D \cup_{\gamma} \epsilon_0DA$.

Considering $\epsilon_0K \equiv \epsilon_0D$, the environment model is constructed by the $\epsilon_0$-NFA of $\epsilon_0K$ after subtracting $\epsilon_0D$. Thus, $\epsilon_0S \triangleq \epsilon_0K \setminus \epsilon_0D$, where the cardinality of $X_S$ is $\theta = \prod_{i=1}^{n} |X_{A_i}|$. The procedure of constructing agents’ and environment models is illustrated in Fig. 1, as well.

![Flow diagram of the operations to produce the models of agents $\epsilon_0A_i$ and the environment $\epsilon_0S$.](image)

**Fig. 1:** Flow diagram of the operations to produce the models of agents $\epsilon_0A_i$ and the environment $\epsilon_0S$.

C. **Task Specification**

Let $x \in X_S$ and let $b$ be an $n$-bit binary indicating the states of the agents that we are interested in specifying. The task specification is a projection $\gamma \triangleq proj(x, b)$ that indicates the desired state of specific agents in the system.

We can now proceed to formally state the following.

D. **Problem statement**

Given a set of individual and inter-agents capabilities and constraints including their failure modes, the initial state of the system and a task specification, determine a string that provides an optimal execution\(^3\) (in terms of total transition cost) that brings the system from any initial state to any state satisfying the task specification.

IV. **FORMULATION AS A MODULE COMPOSITION PROBLEM**

Let us now consider the environment model $\epsilon_0S$ and assume that we would like to use $x_{0,0} \in X_S$ as the initial state of our system. Thus, $S = \Delta(\epsilon_0S, x_{0,0})$. Let $P_{in} \in X_S$ and $P_{out} \in X_S$ 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: $T_j \triangleq \{p_j, e_j, q_j\}$.

Define the $i^{th}$ task module as $T_{0,i} = \{x_{0,0}, e_{0,i}, x_{d,i}\}$ where $proj(x_{d,i}, b) = \gamma$, $x_{d,i} \in X_S$ and $e_{0,i}$ a virtual transition from the initial to the final state. Observe that there are $|\gamma|!$ potential solutions.

To tackle the problem defined in the Problem Statement, we proceed to formulate our problem as a Module Composition Problem (MCP) \cite{18}. Since we are using single-port modules, we will have a special case of the MCP that is solvable in polynomial time. We define $T_i$ as the finite closed module chain containing $T_{0,i}^{-1}$ describing the sequential environment states transitions during the execution defined as: $T_i = \{T_{0,i}^{-1}, T_{1,i}, \ldots, T_{z_i}, i\}$, where $z_i = |T_i|$. In the sequel we will drop the index $i$ for notational brevity.

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 (see \cite{23} for details). 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 \cite{26}, that can be solved utilizing the Dijkstra’s shortest dipath algorithm \cite{27}. The optimal solution composes the $\mathcal{T}$ that minimizes the cost of states transitions in order to fulfill the task specification $\gamma$.

The drawback 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 and is beyond the scope of the current work. 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 \cite{17}, and is currently under active research.

V. **ANALYSIS**

A. **Task Planning Processes**

In this section, we present the properties and process of SPECTER task planner to compute the optimal plan based on the agents’ and environment models. The SPECTER’s running operation phase includes the pre-processing phase and the problem solving phase.

The pre-processing phase includes the construction of the agents models, the environment model (see Fig.1) and the graph of $\epsilon_0S$ using adjacency matrix representation. The computational time to construct $\epsilon_0A_i$ is $O(|X_{A_i}|^2)$ and $O(|X_S|^3)$ for $\epsilon_0S$. It should be noted that $\epsilon_0S$ is only needed to be constructed once and can be used thereafter for all possible task specifications and initial states. $\epsilon_0S$ is converted to a weighted directed graph $\mathcal{H}_S = (V_S, E_S)$, where the set of nodes $V_S$ corresponds to the set of states $X_S$, the set of edges $E_S$ is defined by $f_S$ associated with its cost $g_S$. The adjacency matrix representation of $\mathcal{H}_S$ is a 2-dimensional array $X_S \times X_S$. Each element in the array stores the cost $g_S$ related to the edge $f_S(x \in X_S, e \in E_S)$. The amount of space required to store the array is $O(|V_S|^2)$ in worst case. Computational efficiency can be succeeded if the pre-processing computations are made beforehand. The constructed environment model can then be used to solve all the possible task planning problems by arbitrarily choosing the initial and target states (projection) without the need to reconstruct the environment model.
The problem solving phase encapsulates the synthesis of the optimal task plan and the integration of individual agent failure mode. The Dijkstra's algorithm is implemented over the weighted graph $\mathcal{H}_S$ to find the optimal task plan $\mathcal{T}$ as given in Algorithm 1. In the case where a fault is detected for the transition from state $x_i$ to $x_j$ of $S$ (e.g., by some fault identification system) that affects agent $\nu$, 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 $x'_j \in X'_j$. The above procedures incorporate on-the-fly a possible new failure mode into $S$ without the need to repeat the costly pre-processing phase. The computational time required for this modification is $\theta' = \prod_{i=1}^{N-1} i \neq \nu |X_{A_i}|$ in the worst case.

The task planning problem of minimizing the length of the plan is NP-hard [28]. Let $P$ be the solution to this problem found after running Dijkstra’s algorithm and let $P_i$ denote it’s $i$’th element, $i \in \{1, \ldots |P|\}$. In Algorithm 2, the optimal solution can then be found by running the algorithm for all states that satisfy the task specification, that is $\prod_{i=1}^{N-1} |X_{A_i}|$ times in the worst case scenario, where $\sigma$ denotes the agent index that was used for the task specification $\gamma$. The running time of Dijkstra’s algorithm implementing the Complete Function of Algorithm 1 is $O((|V_S| + |E_S|) \log |V_S|)$.

Algorithm 1 Create module chain $T$.

```
Require: $x_0$-NFA $V_S$, initial state $x_0, S$, task specification $\gamma$, solution type $ST$ (“Complete” or “Heuristic”)
Ensure: Module chain $T$
1: Initialize $\mathcal{H}$ to be a zero matrix, $E$ to be an empty cell array and $T$ to an empty set
2: $S \leftarrow \Delta^{\forall \nu S, x_0, S}$
3: if $proj(x_0, S, b) = \gamma \wedge x_0, S \in X_m, S$ then
4: return $T$
5: else
6: for $i \in \{1, \ldots |X_S|\}$ do
7: for $j \in \{1, \ldots |X_S|\}$ do
8: if $\exists e \in E_S : f_S(x_i, e) = x_j$ then
9: $\mathcal{H}[i][j] \leftarrow g_S(e)$
10: $E[1][j] \leftarrow e$
11: end if
12: end for
13: end for
14: if $ST = "Complete"$ then
15: $[P, cost] \leftarrow \text{COMPLETE}(\mathcal{H}, S, x_0, S, \gamma)$
16: else
17: $[P, cost] \leftarrow \text{HEURISTIC}(\mathcal{H}, S, E, x_0, S, \gamma)$
18: end if
19: for $i \in \{1, \ldots |P| - 1\}$ do
20: $T \leftarrow T \cup \{P_i, E[P_i] \{P_{i+1}\}, P_{i+1}\}$
21: end for
22: return $T$, cost
23: end if
```

While this might be feasible for problems of relatively small size, a heuristic approach is proposed to reduce the computational time required to find a sub-optimal solution. The proposed heuristic, implemented in Algorithm 3, maintains the complexity to the one of Dijkstra's algorithm, while sacrificing optimality and completeness. The computational time of the proposed heuristic is $O(|E_S| \log |V_S|)$. We track the solution to the minimum element $k$ where $proj(P_k, b) = \gamma$ and use the solution $P^* = \{P_1, \ldots P_k\}$. While there are some cases where the proposed heuristic recovers optimality, a classification of those cases and the conditions for feasibility of solutions is currently under consideration but is not part of the current work. The intuition here is to set the goal state $x_d$ to be such that $proj(x_d, b) = \gamma$ and $proj(x_d, b) = proj(x_0, S, b)$ where $\bar{b}$ is the bitwise negation.

Algorithm 2 Complete Function.

```
1: function COMPLETE($\mathcal{H}, S, x_0, S, \gamma$)
2: Initialize $\text{cost}_{\text{min}} \leftarrow \infty$
3: for $i \in \{1, \ldots |X_S|\}$ do
4: for $j \in \{1, \ldots |X_S|\}$ do
5: if $proj(x_i, b_j) = \gamma \wedge x_j \in X_m, S$ then
6: $[P, \text{cost}] \leftarrow \text{Dijkstra}(\mathcal{H}, x_0, x_j)$
7: if $P = \emptyset$ then
8: return Task infeasible.
9: end if
10: if $\text{cost} < \text{cost}_{\text{min}}$ then
11: $\text{cost}_{\text{min}} \leftarrow \text{cost}$
12: $P_{\text{optimal}} \leftarrow P$
13: end if
14: end if
15: end for
16: end for
17: return $P_{\text{optimal}}, \text{cost}_{\text{min}}$
18: end function
```

Algorithm 3 Heuristic Function.

```
1: function HEURISTIC($\mathcal{H}, S, E, x_0, S, \gamma$)
2: Initialize $x_d$ to an empty set
3: Find $x_d \in X_m, S : proj(x_d, b) = \gamma \wedge proj(x_d, \bar{b}) = proj(x_0, S, \bar{b})$
4: Initialize $c^* \leftarrow 0$
5: $[P, \text{cost}] \leftarrow \text{Dijkstra}(\mathcal{H}, x_0, x_d)$
6: if $P = \emptyset$ then
7: return No path to $x_d$.
8: end if
9: for $k \in \{2, \ldots |P|\}$ do
10: $c^* \leftarrow c^* + g_S(E[P_{k-1}] \{P_k\})$
11: if $proj(P_k, b) = \gamma$ then
12: $P^* \leftarrow \{P_1, \ldots P_k\}$
13: return $P^*, c^*$
14: end if
15: end for
16: end function
```

B. Completeness and Optimality

We have the following results regarding the completeness of Algorithms 1 and 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 III, the resulting environment model 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.** Please see the Proof of the same in [23] \(\Box\)

With the completeness property in place we can now state the following optimality and completeness result:
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 III, 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 to achieve task specification γ.

Proof. Please see the Proof of the same in [23] □

VI. CASE STUDY

The proposed methodology could be applied in various applications, such as manufacturing logistics. To demonstrate the applicability of the proposed methodology, several case studies were carried out and one of them is presented below. Additional case studies can be found in [23].

In this case study, we identified a pattern of two regular workflows concerning the manufacturing of semi-finished products 1 and 2 and final products 1. The construction of semi-finished products 1 requires raw materials 1 and 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 stations 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 station F and then are transported to Warehouse.

To model the two workflows requires to define the agents involved in the workflow processes. Considering that 9 agents are involved where $\epsilon A_1, \epsilon A_2, \epsilon A_3, \epsilon A_4, \epsilon A_5$ model the raw materials 1, 2 and 3, $\epsilon A_6$ model the semi-finished products 1 and 2, $\epsilon A_7$ models the final product 1, $\epsilon A_2, \epsilon A_8$ model robots 1 and 2, and $\epsilon A_9$ models the human-worker. The initial states of the agents are: Raw materials 1, 2 and 3 are stored in the Warehouse, each robot is at its docking station and the human is in Warehouse. The objective tasks are: (a) produce $\epsilon A_6$ and store it in Warehouse and (b) produce $\epsilon A_4$ and $\epsilon A_5$, both in minimum time. Models of $\epsilon A_7, \epsilon A_8$ and $\epsilon A_9$ are constructed by combing the agents capabilities and constraints. Capabilities and constraints of $\epsilon A_1, \epsilon A_2, \epsilon A_3, \epsilon A_4, \epsilon A_5, \epsilon A_6$ are defined as inter-agent capabilities and inter-agent capabilities constraints. Thus, $\epsilon S$ is constructed, where $X_S = \prod_{n=1}^{6} |X_{A_n}| = 1.875 \times 10^6$ states, with $|X_{A_1}| = |X_{A_2}| = |X_{A_3}| = |X_{A_4}| = |X_{A_5}| = |X_{A_6}| = 5$ states, $|X_{A_7}| = 6$ states, and $|X_{A_9}| = 4$ states. The task specification is defined as $proj(x_d, b) = JCW$, with $b = 0001110000$ and $|\gamma| = 3$.

The solution $T$ was computed with both Algorithms 2 and 3. Both solutions were identical. The computed $T$ resulted to a composition of 24 open chain modules. In words, $(T_1)$ raw materials 1, 2, 3 are stored in $W$, 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 2, $(T_7)$ robot 1 goes to injection 2 while carrying raw materials, $(T_8)$ worker unloads raw material 3 from the robot 1 and loads the raw material 3 to the C, $(T_9)$ robot 1 goes to injection 2 while carrying raw materials 1 and 2, $(T_{10})$ worker goes to injection 2, $(T_{11}) C$ starts the production of semi-finished product 2, $(T_{12})$ worker unloads raw material 2 from robots 1 and loads the raw material 2 to J, $(T_{13})$ injection machine starts the preparation of semi-finished product 1, $(T_{14})$ worker unloads raw material 1 from robot 1 and loads the raw material 1 to J, $(T_{15})$ injection 1 starts the production of semi-finished product 1, $(T_{16})$ the final product 1 is produced at Fa, $(T_{17})$ semi-product 2 is produced by C, $(T_{18})$ worker goes to Fa, $(T_{19})$ robot 2 goes to Fa, $(T_{20})$ worker loads final product 1 on robot 2, $(T_{21})$ robot 2 goes to W, $(T_{22})$ semi-finished product 1 is produced by J, $(T_{23})$ worker goes to W, $(T_{24})$ worker unloads the final product 1 from robot 2 to W.

The pre-processing runtime to construct the agents and environment models is $3.081 \times 10^5 s$. The problem solving runtime to find $T$ utilizing Algorithm 3 is $165.73 s$ whereas with Algorithm 2 it required a total of $10,720 s$ for all 15,000 possible $x_d$’s for which $proj(x_d, b) = JCW$.

VII. CONCLUSIONS

The methodology for a novel multi-agent supervisory control task planner is proposed. Given the capabilities, constraints and failure modes of the agents under the framework of NFAs with $\epsilon$-transitions, SPECTER produces the optimal sequence of tasks for transporting the state of the environment from any initial to any given destination state. The developed algorithms can provide a complete solution with optimality guarantees whenever a solution exists. By relaxing the completeness property requirement, a significant reduction in the computational requirements is achieved, that provides suboptimal solutions through the use of efficient heuristics. The results of the case study demonstrate the applicability as well as the effectiveness and 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. Future work will focus on combining SPECTER with supervisory control theory to enable reactive execution in dynamic environments.
