Safe Learning of Multi-Agent Action Models from Concurrent Joint Action Observations

ARGAMAN MORDOCH, Ben-Gurion University of the Negev, Israel
ORI KARAT, Ben-Gurion University of the Negev, Israel
LEA SHMILOVICH, Ben-Gurion University of the Negev, Israel
YARIN BENYAMIN, Ben-Gurion University of the Negev, Israel
BRENDAN JUBA, Washington University in St. Louis, USA
RONI STERN*, Ben-Gurion University of the Negev, Israel

Background: Multi-Agent Planning (MAP) involves coordinating the actions of multiple autonomous agents to achieve shared objectives. A prevalent formalism for MAP is the Multi-Agent Planning Domain Definition Language (MA-PDDL). While effective, existing MA-PDDL solvers typically require complete access to agents' action models—specifically their preconditions and effects. However, manually creating these models is often intractable, requiring exhaustive domain expertise.
Objectives: This work explores an alternative approach: automatically learning agents' action models from observed transitions. Since learned models may be inaccurate, planning with them can yield invalid or non-executable sequences. To mitigate this, we formalize a requirement for *safety*, ensuring that plans generated via the learned model remain sound with respect to the real unknown action model.
Methods: Previous research introduced the Safe Action Model Learning (SAM) algorithm for single-agent domains. However, SAM is not suitable for MA-PDDL environments where observations include concurrently executed actions, since it cannot naturally disambiguate the individual contributions of the agents to the observed effects. To address this, we introduce Multi-Agent Safe Action Model Learning (MA-SAM), a safe action model learning algorithm designed to handle concurrent multi-agent observations. For scenarios where individual action effects remain ambiguous, we further propose MA-SAM$^+$, which learns the preconditions and effects of *macro-actions* representing concurrent execution of subsets of actions. We evaluate both algorithms on domains from the Competition of Distributed and Multi-Agent Planners (CoDMAP) benchmarks and a novel MAP domain inspired by the game *Overcooked*.
Results: We establish a theoretical lower bound on the sample complexity for learning safe action models in multi-agent settings. We prove that MA-SAM does not achieve this lower bound in all cases, identifying specific conditions under which its sample complexity may become unbounded. Empirically, both MA-SAM and MA-SAM$^+$ significantly outperform SAM-based baselines in coverage and applicability rates. While their performance is comparable in many settings, MA-SAM$^+$ highly outperforms MA-SAM in some of the evaluated domains.
Conclusions: We present the first algorithms capable of learning safe MA-PDDL action models from concurrently executed actions, providing both theoretical foundations and empirical validation across diverse planning benchmarks.

JAIR Associate Editor: Siddharth Srivastava

*Corresponding Author.

Authors' Contact Information: Argaman Mordoch, ORCID: 0000-0003-3502-1461, mordocha@post.bgu.ac.il, Ben-Gurion University of the Negev, Be'er Sheva, Israel; Ori Karat, karato@post.bgu.ac.il, Ben-Gurion University of the Negev, Be'er Sheva, Israel; Lea Shmilovich, leashmilovich@gmail.com, Ben-Gurion University of the Negev, Be'er Sheva, Israel; Yarin Benyamin, ORCID: 0009-0008-7427-6380, bnyamin@post.bgu.ac.il, Ben-Gurion University of the Negev, Be'er Sheva, Israel; Brendan Juba, ORCID: 0000-0001-6542-833X, bjuba@wustl.edu, Washington University in St. Louis, St. Louis, Missouri , USA; Roni Stern, ORCID: 0000-0003-0043-8179, roni.stern@gmail.com, Ben-Gurion University of the Negev, Be'er Sheva, Israel.

DOI: 10.1613/jair.1.20494

JAIR Reference Format:
Argaman Mordoch, Ori Karat, Lea Shmilovich, Yarin Benyamin, Brendan Juba, and Roni Stern. 2026. Safe Learning of Multi-Agent Action Models from Concurrent Joint Action Observations. *Journal of Artificial Intelligence Research* 86, Article 1 (May 2026), 35 pages. DOI: 10.1613/jair.1.20494

1 Introduction

Many real-world applications involve a centralized controller planning for multiple agents, e.g., in furniture assembly systems (Knepper et al. 2013), automated warehouses (Azadeh et al. 2019), airport ground handling management (Kabongo et al. 2016), and disaster response (Ramchurn et al. 2016). The main paradigms to plan and coordinate multiple agents are *Multi-Agent Reinforcement Learning (MARL)* (Tan 1993) and *Multi-Agent Planning (MAP)* (R. I. Brafman and Domshlak 2013; Torreno et al. 2017). MARL methods use trial-and-error approaches to learn *policies* for the agents that maximize a cumulative reward function. These methods often require many interactions with the environment before finding sufficiently effective policies. MAP methods do not require any interactions with the environment. Instead, they plan by predicting the outcome of interactions with the environment using an *action model*, which is a formal description of the preconditions and effects of the agents' actions. Based on the correctness of the action model, MAP methods can also provide formal soundness, completeness, and optimality guarantees in some settings.

However, such action models are often not available. One reason for this is that creating them manually is difficult and time consuming. This can be particularly problematic in *ad-hoc teamwork* setups (Barrett and Stone 2012), where a team of agents is formed for a specific mission. In such cases, the agents are often different from each other, and thus have different action models. However, the agents' interfaces may not support sharing their action models (Verma et al. 2021). **In this work, we explore the problem of automatically learning an action model for multiple agents by observing the behavior of these agents in the domain.**

Specifically, we focus on learning action models for MAP problems that can be represented using the Multi-Agent Planning Domain Definition Language (MA-PDDL) (R. I. Brafman and Domshlak 2013; Kovacs 2012). MA-PDDL is an extension of the well-studied PDDL formalism (McDermott et al. 1998) for defining single-agent planning problems in deterministic and discrete environments. Centralized and decentralized MAP algorithms have been proposed for solving problems defined in MA-PDDL (Komenda et al. 2016; Maliah et al. 2016a, 2014; Torreno et al. 2017). Yet the topic of learning an MA-PDDL action model from observations has rarely been studied before.

In contrast, learning single agent PDDL action models from observations has been extensively studied (Aineto, Jiménez, et al. 2022; Amir and Chang 2008; S. N. Cresswell et al. 2013; Juba, Le, et al. 2021; Zhuo and Kambhampati 2013, inter alia.) Zhuo, Muñoz-Avila, et al. (2011) generalized the ARMS algorithm (K. Wu et al. 2007) to learn a MA-PDDL action model. However, they assumed the given observations are *sequential* plan traces. That is, agents are never observed acting concurrently. This is a significant limitation since in many multi-agent systems agents act concurrently. In this work, we bridge this gap and explore how to learn MA-PDDL action models (R. I. Brafman and Domshlak 2013) from observations that include agents acting *concurrently*. Note that even if agents' actions are not executed exactly at the same time, they are effectively concurrent if we only observe the state before and after their execution.

A baseline approach to address this learning problem is to ignore observations that include concurrent actions. This approach is inefficient since it wastes valuable observations. Another baseline approach is to consider every set of concurrently executed actions as a separate action. This is the only safe solution without making any assumption about the interactions between the agents' actions. However, it fails to generalize beyond the observed combinations of actions it has seen being executed concurrently. One may even say, that learning is of interest if some structure or regularity can be leveraged to generalize beyond a small number of observations. In this work, we assume such structure exists. Specifically, we make the common assumption in MA-PDDL that

concurrent actions do not obstruct or alter the outcomes of other agents' actions (Crosby, Jonsson, et al. 2014; Komenda et al. 2016) . More challenging assumptions about the interactions between concurrently executed actions have rarely been considered (Boutilier and R. I. Brafman 2001; Shekhar and R. I. Brafman 2020) and most MA-PDDL planners do not support them.[1] Even under this assumption, learning from observations of multiple agents acting concurrently is still challenging since it may be unclear which of the agents' actions is responsible for an observed effect. For example, if agents 1 and 2 perform actions a_1 and a_2, and in the next state a new fluent f appears, it is unclear if f is an effect of a_1, a_2, or both. To address this learning challenge, we propose the MA-SAM algorithm.

MA-SAM is a generalization of the Safe Action Model Learning (SAM) learning algorithm (Juba, Le, et al. 2021). MA-SAM learns an action model for each of the agents by iterating over the given observations and applying logical inference rules to determine the preconditions and effects of the observed actions. We theoretically analyze MA-SAM, showing that any multi-agent plan created with the action models it learns is guaranteed to execute precisely as predicted by the learned models (Theorem 4.4). Action models with this property are called *safe* (Juba, Le, et al. 2021; Stern and Juba 2017). Having a safe action model allows for enforcing other notions of safety, such as ensuring that a safety function over the current state never drops below some predefined threshold (Wachi et al. 2018). In addition, we show that MA-SAM runs in polynomial time (Theorem 4.5), and provide a lower bound on its sample complexity (Theorem 4.8). We show that the lower bound sample complexity is polynomial in the number of fluents and actions, but exponential in the number of agents. A special case of our lower bound complexity calculations is when there is only one agent in the environment, i.e., single-agent PDDL. This novel result shows that the previously proposed SAM algorithm has an optimal sample complexity.

In some cases, however, MA-SAM cannot discern which action created a specific effect, e.g., when some actions are only observed to be executed concurrently. To resolve such ambiguities, we propose an extension to MA-SAM, named MA-SAM$^+$. MA-SAM$^+$ creates *Lifted Macro Actions* that represent the concurrent execution of a subset of actions.[2] We show that the MA-PDDL action models learned by MA-SAM$^+$ are safe and that MA-SAM$^+$'s runtime complexity is exponential in the number of agents, and polynomial in the number of actions, parameter-bound literals, and observations.

We implemented MA-SAM and MA-SAM$^+$, and evaluated them experimentally on standard MA-PDDL benchmarks, as well as on the novel domain we introduce that is based on the popular Overcooked video game. As a baseline, we used a version of the SAM learning algorithm that ignores observations with concurrent actions. The primary performance metric we use is *coverage*, i.e., the percentage of test set problems solved when using an off-the-shelf planner given the action model learned by the evaluated algorithm. In addition, we assess the quality of the learned models in terms of action applicability, as well as syntactic precision and recall. The results show that MA-SAM$^+$ and MA-SAM perform the same or better than SAM across all evaluated performance metrics. As expected, the advantage of MA-SAM$^+$ and MA-SAM is most evident in settings where more actions are observed to be executed concurrently. Furthermore, in cases where MA-SAM could not determine which agent's action resulted in the observed effects, MA-SAM$^+$ demonstrated a significant performance advantage.

Finally, we discuss approaches to learning more complex MAP action models in settings where agents' actions may interfere with one another or exhibit unique behavior when executed concurrently (Section 7).

2 Background and Related Work

The Planning Domain Definition Language (PDDL) (McDermott et al. 1998) is the de-facto standard for defining single-agent classical planning problems. *Multi-Agent PDDL* (R. I. Brafman and Domshlak 2013; Kovacs 2012) is

[1] Nevertheless, we discuss such assumptions in Section 7.

[2] These macro actions are different from macro actions in single-agent planning, which are *sequences* of actions.

an extension of PDDL to support planning for multiple agents.[3] A multi-agent planning problem in MA-PDDL is defined by two components: a MA-PDDL *domain* and a MA-PDDL *problem*.

Definition 2.1 (MA-PDDL Domain). An MA-PDDL domain is defined by the tuple:

$$D = \left\langle Types, F, k, \{A_i\}_{i=1}^{k}, \{M_i\}_{i=1}^{k} \right\rangle$$

where $Types$ is a set of symbols representing types of objects, F is a set of lifted fluents, k is the number of agents, A_i are the lifted actions agent i can perform, and M_i is its action model.

A *lifted fluent* f is a pair $\langle name(\mathrm{f}), params(\mathrm{f})\rangle$ where $name(\mathrm{f})$ is a symbol and $params(\mathrm{f})$ represent a relation over typed *objects*. We denote by $type(o)$, the type $type \in Types$ of an object o. A *binding* of a lifted fluent f is a function mapping every parameter of f to an object of the indicated type. A *grounded fluent* f is a pair $\langle \mathrm{f}, b_{\mathrm{f}}\rangle$ where f is a lifted fluent and b_{f} is a binding for f. The term *literal* refers to either a fluent or its negation. The definitions of binding, lifted, and grounded fluents transfer naturally to literals. A *state* of the world is a set of grounded literals that includes, for every grounded fluent f, either f or $\neg f$.

A *lifted action* α is a pair $\langle name(\alpha), params(\alpha)\rangle$ where $name(\alpha)$ is a symbol, $params(\alpha)$ is a list of types. For each agent i, its action model M_i is defined as a pair of functions pre_{M_i} and eff_{M_i} that map agent i's actions to their preconditions and effects, respectively. Note that agents may have different lifted actions and different action models. Preconditions and effects are conjunctions of *parameter-bound literals* (pb-literals). A pb-literal for a lifted action α is a pair $\left\langle \ell, b_{\ell,\alpha}\right\rangle$ where ℓ is a lifted literal and $b_{\ell,\alpha}$ is a function that maps each parameter of ℓ to a parameter in α. We denote M as the set of all the agents action models, i.e., $M = \{M_i\}_{i=1}^{k}$. For convenience, we denote by $pre_M(\alpha)$ and $eff_M(\alpha)$ the preconditions and effects of action α, respectively. A *grounded action* a is a tuple $\langle \alpha, b_\alpha\rangle$ where α is a lifted action and b_α is a binding of α. A grounded action a is *applicable* in a state s, denoted as $app_M(a, s)$, if every grounded literal $p \in pre_M(a)$ is in s. The result of applying a to a state s, denoted by $a(s)$, is a state composed of the fluents in $eff_M(a)$ and the fluents in s except those whose negations are in $eff_M(a)$.

Definition 2.2 (MA-PDDL Problem). An MA-PDDL problem is defined by a tuple $\pi = \langle O, I, G\rangle$ where O is the set of typed objects, I is the initial state, and G is the desired goal.

A plan π is a sequence of grounded actions $a_1, a_2, ..., a_n$ such that (1) a_1 is applicable in the initial state I, (2) for each $1 \leq j \leq n$, a_j is applicable in the state $a_{j-1}(...a_1(I)...)$, and (3) $G \subseteq a_n(a_{n-1}(...a_1(I)...))$.

Example 2.3. Consider an e-commerce platform with multiple users, where customers perform actions such as *searching* for products, *rating* them, or *buying* them. We wish to design an agent with the same capabilities as a customer. This is useful for testing purposes, as well as for designing helpful customer-assistance bots. In our example, the possible actions a customer can perform are defined by the following lifted actions: $(buy\text{-}item\ ?a - customer\ ?i - item)$, $(rate\text{-}item\ ?a - customer\ ?i - item)$, $(search\ ?a - customer\ ?i - item\ ?c - category)$ and $(browse\text{-}item\ ?a - customer\ ?i - item)$. In this example, the *customer* is the agent, and *item* and *category* are types of objects. Various fluents exist to represent a state, such as $(item\text{-}in\text{-}cart\ ?a - customer\ ?i - item)$, representing that item $?i$ is in customer $?a$'s cart; $(item\text{-}rated\ ?a - customer\ ?i - item)$ representing that the customer $?a$ rated the item $?i$, $(item\text{-}searched\ ?i - item\ ?c - category)$ representing that the item $?i$ was searched under the category $?c$; and $(item\text{-}browsed\ ?i - item)$ representing that the item's information and details were browsed.

Some works on MA-PDDL considered aspects such as privacy between the agents and decentralized planning and execution. In this work, we focus on a *centralized* planning setting and assume that agents have no private actions.

[3] PDDL can be viewed as the parameterized version of STRIPS (Fikes and Nilsson 1971), and similarly, MA-PDDL is the parameterized version of MA-STRIPS (R. I. Brafman and Domshlak 2013).

2.1 Solving MA-PDDL

Multiple MA-PDDL planners have been previously proposed, such as CSP+Planning (R. I. Brafman and Domshlak 2013), MAFS (Nissim and R. Brafman 2014), GPPP (Maliah et al. 2014), DPP (Maliah et al. 2016b), and PSM (Tožička, Jakubův, et al. 2016). The CSP+Planning (R. I. Brafman and Domshlak 2013) algorithm coordinates between the agents by assuming each agent has a set of at most δ coordination points. A coordination point is a point during the plan where the agents need to interact with one another. It integrates planning with Distributed Constraint Satisfaction Problem (DisCSP) solving to ensure both the validity of the agents' interactions and the consistency of each agent's individual plan projection. MAFS (Nissim and R. Brafman 2014) is a distributed state-space search algorithm in which each agent performs only part of the state expansions relevant to it. The algorithm offers similar flexibility as presented in single-agent forwards-search planning algorithms. The Greedy Privacy-Preserving Planner (GPPP) (Maliah et al. 2014) applies a global planning stage and then a local planning stage. In the global planning stage, agents agree on a joint coordination scheme by solving a relaxed MAP task that only contains public actions. In the local stage, agents compute private plans to achieve the preconditions of the actions in the skeleton plan. The DPP (Maliah et al. 2016b) algorithm uses projections that maintain the dependencies between the agents' public actions to speed up planning. Planning State Machines (PSM) (Tožička, Jakubův, et al. 2016) uses a set of finite automata (PSMs), where each automaton represents a set of local plans of a given agent containing private and public facts. The agents exchange public projections of their PSMs until a solution is found. For a survey on MA-PDDL algorithms, see (Torreno et al. 2017).

Some assume that the planning task is *centralized* (Borrajo 2013; Crosby, Rovatsos, et al. 2013), i.e., there exists a single controller agent that is responsible for decision-making and coordination between the agents. Other MA-PDDL planners assume that the planning task is decentralized (Nissim and R. Brafman 2014). Various forms of privacy and corresponding algorithms have also been discussed in the context of MA-STRIPS (R. I. Brafman 2015; Maliah et al. 2016b; Tožička, Štolba, et al. 2017), varying by the information they allow agents to know and infer about each other.

The above planning algorithms assume that the planning action model is available before executing the planning process. This assumption may not hold in some cases. To address these cases, many action model learning approaches were created throughout the years. Next, we present some prominent examples of action model learning algorithms.

2.2 Action Model Learning

Different algorithms have been proposed for learning planning action models for *single-agent planning*. Most action model learning algorithms learn action models by processing a set of *trajectories*, which are given as input. A *trajectory* $T = \langle s_0, a_1, s_1, \ldots a_n, s_n \rangle$ is an alternating sequence of states $(s_0, \ldots, s_n)$ and actions $(a_1, \ldots, a_n)$ that starts and ends with a state, where $s_i = a_i(s_{i-1})$ for $i = 1, \ldots, n$. Some action model learning algorithms require access to all the actions and states in the given trajectories, while others can handle missing states and actions. For example, the ARMS algorithm (Yang et al. 2007) requires plan examples, where each plan example consists of the initial and goal states, and the sequence of grounded actions that lead from the initial to the goal state. For each observed action, ARMS lifts the action and constructs a set of action and correctness constraints on the fluents involved in its preconditions and effects. These constraints are then resolved using a weighted MAX-SAT solver to generate the action model with the highest weight. The Simultaneous Learning and Filtering (SLAF) (Amir and Chang 2008) learns lifted action models from trajectories even if some of the states in them are not observed. SLAF uses logical inference to filter inconsistent action models, and requires observing all actions in the trajectory. FAMA (Aineto, Celorrio, et al. 2019) can also handle missing observations and outputs a lifted planning domain model. It frames the task of learning an action model as a planning problem, ensuring that the returned action model is consistent with the provided observations. NOLAM (Lamanna and Serafini

2024) can learn lifted action models even from noisy trajectories. LOCM (S. Cresswell and Gregory 2011) and LOCM2 (S. N. Cresswell et al. 2013), analyze only the sequences of actions in the given trajectories, ignoring any information about the states between them. LatPlan (Asai and Fukunaga 2018) and ROSAME-I (Xi et al. 2024) are conceptually different since they learn propositional action models from trajectories where the states in the given trajectories are given as images, as opposed to a conjunction of fluents.

The SAM learning algorithms (Juba, Le, et al. 2021; Juba and Stern 2022; Le et al. 2024; Mordoch, Juba, et al. 2023; Mordoch, Scala, et al. 2024; Stern and Juba 2017) are a family of action model learning algorithms that return action models that guarantee that plans generated with them are applicable in the actual action model. An action model having this property is referred to as a *safe action model*.

Definition 2.4 (Safe Action Model). M' is a safe action model w.r.t. an action model M if for every state s and action a, if a is applicable in s according to M' then (1) a is also applicable in s according to M, and (2) applying a in s results in exactly the same state according to both M and M'. An action model M is called safe if it is safe w.r.t. the real, unknown action model.

The SAM family of algorithms addresses learning safe action models under different settings and assumptions, including lifted action models (Juba, Le, et al. 2021), action models with stochastic (Juba and Stern 2022), action models with conditional effects (Mordoch, Scala, et al. 2024), action models containing numeric preconditions and effects (Mordoch, Juba, et al. 2023), and learning from partially observable traces (Le et al. 2024). However, none of the presented algorithms support learning action models in a multi-agent setting.

To the best of our knowledge, the only algorithm for learning action models in a multi-agent setting is the Lammas algorithm (Zhuo, Muñoz-Avila, et al. 2011), which extends ARMS to output MA-PDDL action models. Lammas augments ARMS by introducing agent constraints that capture potential interactions between agents' actions. To build these constraints, Lammas defines *interactive actions* as those that provide conditions for other agents. It then constructs a weighted Agent Interaction Graph (w-AIG), which specifies which agent's actions produce effects required by others. The w-AIG is subsequently used to derive the agent constraints.

Although Lammas learns MA-PDDL action models, it is unsuitable when the example plans contain concurrent actions. Since edges in the w-AIG are constructed by scanning plan traces for *sequentially* executed interactive actions, the algorithm cannot capture interactions that occur concurrently. As a result, it fails to generate the agent constraints necessary to learn the agents' action models.

As we demonstrate later, learning from trajectories with concurrently executed actions significantly increases the complexity of the problem. To our knowledge, the algorithms introduced in this work are the first to address the challenge of learning action models in a multi-agent setting where agents perform actions concurrently.

3 Problem Definition

We consider a coordinating agent tasked with solving a set of MA-PDDL problems $\Pi = \{\pi_1, \pi_2, ..., \pi_m\}$. The coordinating agent is given access to the agents' domain definition, i.e., $D = \langle Types, F, k, \{A_i\}_{i=1}^{k} \rangle$, however it cannot access the agents' action models. Instead it receives a set of *joint actions trajectories* $\mathcal{T}$. The agent is required to learn a *safe* action model M (Definition 2.4) representing the set of the agents' action models, and use M to solve the problems in Π.

A *joint action* is a k-ary vector $\hat{a}$ where $\hat{a}[i]$ is either an action of the agent i or *NO-OP*. A joint action represents the intention to execute its constituent actions concurrently, where *NO-OP* indicating that the relevant agent does not perform any action. A trajectory of joint actions can be created by executing a sequence π of joint actions starting from a state s. The resulting trajectory is the sequence $(s_0, \hat{a}_1, \ldots, \hat{a}_{|\pi|}, s_{|\pi|})$ where $s_0 = s$ and for all $0 < i \leq |\pi|$, (1) $\hat{a}_i$ is applicable in the state s_{i-1}, i.e., every action $a \in \hat{a}_i$ is applicable in the state s_{i-1} and, (2) $s_i = \hat{a}_i(s_{i-1})$. The state $s_i = \hat{a}_i(s_{i-1})$ is composed of the fluents in $eff_M(\hat{a}[k])$ and the fluents in s_{i-1} except those

whose negations are in $eff_M(\hat{a}[k])$ for all agent k acting in $\hat{a}_i$. Such a trajectory can also be represented as a set of *joint action triplets* $\{ \langle s_{i-1}, \hat{a}_i, s_i \rangle \}_{i=1}^{|\pi|}$.

Example 3.1. Figure 1 presents an illustration of a trajectory of joint actions of three agents, representing customers in an e-commerce platform – $a1$, $a2$ and $a3$. This trajectory consists of two joint action triplets as follows:

(1) $\langle s_0, [(buy\text{-}item\ a1\ item1), (rate\text{-}item\ a2\ item2), (browse\text{-}item\ a3\ item1)], s_1 \rangle$,
(2) $\langle s_1, [(rate\text{-}item\ a1\ item1), (search\ a2\ item3\ c1), (buy\text{-}item\ a3\ item4)], s_2 \rangle$.

The state s_0 includes, among other, the literals (which include both positive and negative fluents) $(not\ (item\text{-}in\text{-}cart\ a1\ item1))$, $(not\ (item\text{-}browsed\ item1))$, $(item\text{-}in\text{-}cart\ a2\ item2)$ and $(not\ (item\text{-}in\text{-}cart\ a3\ item4))$. After executing the first joint action triplet, the state s_1 includes the literals $(item\text{-}in\text{-}cart\ a1\ item1)$, $(item\text{-}browsed\ item1)$, $(item\text{-}rated\ a2\ item2)$. After executing the second joint action, s_2 includes $(item\text{-}rated\ a1\ item1)$, $(item\text{-}searched\ item3\ c1)$ and $(item\text{-}in\text{-}cart\ a3\ item4)$. This example shows the challenge to learning agents' action models from joint action triplets: after observing the first joint action triplet, it is not possible to learn which of the agents' actions has the fluent $(item\text{-}browsed\ item1)$ as an effect.

In real e-commerce applications, actions are unlikely to be executed in perfect synchrony, as small delays between agents may occur. Nevertheless, if a monitoring system samples the environment at fixed intervals, sequential actions occurring within the same interval may appear concurrent. This assumption is reasonable, as aggregating API calls is considerably simpler than sampling the full database of a large-scale application.

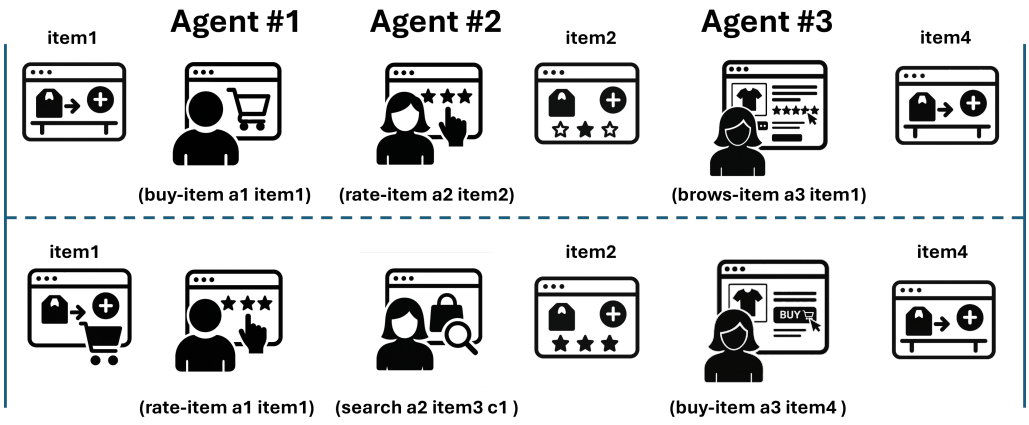

Fig. 1. An illustration of a trajectory of joint actions. The agents $a1$, $a2$ and $a3$ are acting concurrently in our e-commerce application example (Example 2.3). First, customer $a1$ performs the action $(buy\text{-}item\ a1\ item1)$ while customer $a2$ performs action $(rate\text{-}item\ a2\ item2)$, and customer $a3$ performs $(browse\text{-}item\ a3\ item1)$ concurrently. Then, customer $a1$ performs the action $(rate\text{-}item\ a1\ item1)$ while customer $a2$ performs the action $(search\ a2\ item3\ c1)$ and customer $a3$ performs $(buy\text{-}item\ a3\ item4)$ concurrently.

Assumptions. We consider a setting in which the real action model, denoted M^*, is specified in MA-PDDL. This means the agents' actions are deterministic and states are represented as a conjunction of fluents. Preconditions and effects are sets representing conjunctions of pb-literals. That is, we do not support more complex action models, e.g., including conditional effects and disjunctive preconditions. We also assume that the input trajectories

are fully observable, i.e., all the state variables and the executed actions are disclosed to the learning algorithm, and are created by executing sound plans for other problems in the domain. Full observability here also means knowing which agent performed which action. This setting is common in real-world settings: the API of the agents is known but their exact functionality, i.e., when the API can be used (preconditions) and what it does (effects), is not known, or we do not have a formal model of it. For example, consider monitoring logs of a web application. In these logs, the executed API requests can be explicitly seen with their headers and payload data (as long as they conform with user privacy settings). In these headers, the users making the API requests can be identified by unique IDs – many analysis companies use this data to improve web services. Furthermore, knowing which agent performed which action is standard in Multi-Agent Reinforcement Learning (MARL) when using the very common Centralized Training Decentralized Execution (CTDE) setting. For simplicity, we also assume that every action in a joint action in the input trajectories consists of a grounded action $a = \langle \alpha, b_\alpha \rangle$ in which the binding b_α is an injective function. This means each parameter of α is mapped to a distinct object. Generalizing to a setting where this assumption, known as the *injective binding assumption*, does not hold, has been discussed in prior work (Juba, Le, et al. 2021).

The MA-PDDL model does not support specifying *collaborative actions*, i.e., sets of actions, executed concurrently, whose effects differ from the union of their constituent single-agent actions. Consequently, throughout this paper, we assume that concurrently executed actions are *well-defined* (Crosby, Jonsson, et al. 2014). A joint action is called *well-defined* if (1) the conjunction of its constituent actions' preconditions is satisfiable, and (2) the conjunction of its constituent actions' effects are satisfiable. Additionally, we require that no action's effects contradict the other actions' preconditions. Extensions to MA-PDDL that allow more complex interactions between agents' actions have been proposed (Shekhar and R. I. Brafman 2020). In Section 7, we discuss cases where these assumptions do not hold.

Under these assumptions, a multi-agent action model M for k agents can be represented as k single-agent action models $\{M_i\}_{i=1}^k$. The following observation captures the relationship between the safety property of a multi-agent action model and that of its constituent single-agent action models.

Observation 1. *A multi-agent action model $M' = \{M'_i\}_{i=1}^k$ is safe w.r.t $M = \{M_i\}_{i=1}^k$ iff M'_i is safe w.r.t M_i for all i.*

Proof. If M is safe w.r.t M', then it must satisfy the following equation for each state s:

$$app_{M'}(a, s) \rightarrow \Big(app_M(a, s) \wedge apply(a, s, M') = apply(a, s, M) \Big) \tag{1}$$

where $apply(a, s, M)$ is the application of the grounded action a in s according to the action model M.

The above holds for every joint action, including joint actions in which only one agent acts and the rest of the actions are *NO-OP*. Thus, the safety of the single-agent action models is guaranteed. If the single-agent action models are safe, then for every joint action, the union of the preconditions of its constituent single-agent action according to M' is a (possibly not strict) superset of the preconditions according to M. A similar argument applies to the effects as well. □

4 Multi-Agent SAM (MA-SAM)

Next, we present our first contribution: the MA-SAM algorithm for learning safe action models from trajectories including joint action triplets. To this end, we introduce the following notation and learning rules. For a grounded action $a = \langle \alpha, b_\alpha \rangle$ and a grounded literal $l = \langle \ell, b_\ell \rangle$, we denote by $objects(a)$ and $objects(l)$ the set of objects bound to the parameters of a and the set of objects bound to the parameters of l, respectively. We say that a grounded action a is *relevant* to a grounded literal l if $objects(l) \subseteq objects(a)$. For example, the grounded action *(buy-itema1 item1)* from our e-commerce example, is relevant to the literal *(item-in-carta1 item1)*. Note that the

pb-literal $\langle \ell, b_{\ell,\alpha} \rangle$ can be a precondition or an effect of the lifted action α only if the grounded action a is relevant to the grounded literal l. We denote by $relevant(l, \hat{a})$ the set of all grounded actions in the joint action $\hat{a}$ that are relevant to l.

Observation 2 (MA-SAM Rules for Lifted Actions). *For any observed joint action triplet $\langle s, \hat{a}, s' \rangle$*
(1) If $l \notin s$, where $l = \langle \ell, b_\ell \rangle$, then for every $\langle \alpha', b_{\alpha'} \rangle \in relevant(l, \hat{a})$ it holds that

$$\langle \ell, b_{\ell,\alpha'} \rangle \textbf{ is not a precondition of } \alpha'$$

(2) If $l \notin s'$, where $l = \langle \ell, b_\ell \rangle$, then for every $\langle \alpha', b_{\alpha'} \rangle \in relevant(l, \hat{a})$ it holds that

$$\langle \ell, b_{\ell,\alpha'} \rangle \textbf{ is not an effect of } \alpha'$$

(3) If $l \in s' \setminus s$, where $l = \langle \ell, b_\ell \rangle$, then there exists $\langle \alpha', b_{\alpha'} \rangle \in relevant(l, \hat{a})$ for which

$$\langle \ell, b_{\ell,\alpha'} \rangle \textbf{ is an effect of } \alpha'$$

Informally, rule 1 states that for every grounded literal l unsatisfied in a state immediately before $\hat{a}$, and for every lifted action α' such that its grounding is in $relevant(l, \hat{a})$, the pb-literal $\langle \ell, b_{\ell,\alpha'} \rangle$ cannot be its precondition. Similarly, rule 2 states that for every grounded literal l unsatisfied in a state immediately after $\hat{a}$, and for every lifted action α' such that its grounding is in $relevant(l, \hat{a})$, the pb-literal $\langle \ell, b_{\ell,\alpha'} \rangle$ cannot be its effect. Rule 3 is slightly different, stating that for any grounded literal l that is unsatisfied in the state immediately before $\hat{a}$ is applied and satisfied in the state immediately after, there exists at least one lifted action α' such that its grounded execution is in $relevant(l, \hat{a})$ and $\langle \ell, b_{\ell,\alpha'} \rangle$ is its effect.

The rules in Observation 2 generalize the inductive rules used by SAM (Juba, Le, et al. 2021) to trajectories including joint action triplets. Our MA-SAM algorithm (Algorithm 1) uses these rules to learn a safe action model as follows:

Learning Preconditions. MA-SAM initially sets the preconditions of every lifted single-agent action to include every possible pb-literal (line 3 in Alg. 1).[4] Then, for every observed joint action triplet $\langle s, \hat{a}, s' \rangle$ and every lifted single-agent action α such that $\langle \alpha, b_\alpha \rangle = a \in \hat{a}$, MA-SAM removes all pb-literals that cannot be preconditions due to Rule 1 in Obs. 2 (line 10). The preconditions for every lifted action α are thus always a (possibly not strict) superset of the preconditions for α in M^*.

Learning Effects. MA-SAM creates for every lifted literal ℓ a Conjunctive Normal Form (CNF) formula, denoted CNF_ℓ, describing constraints we learn about the lifted actions that may have ℓ as an effect. The atoms of CNF_ℓ are of the form $\text{IsEff}(\langle \ell, b_{\ell,\alpha} \rangle, \alpha)$ representing that the pb-literal $\langle \ell, b_{\ell,\alpha} \rangle$ is an effect of the lifted action α. These CNFs are initially empty (line 4). Then, for every observed joint action triplet $\langle s, \hat{a}, s' \rangle$ MA-SAM adds constraints by applying Rules 2 and 3 in Obs. 2 (lines 19 and 14). After processing all the action triplets, MA-SAM minimizes the resulting CNFs by applying unit propagation. MA-SAM then iterates over these CNFs and identifies the set of actions that can be *safely applied*, which is denoted A_{safe} and defined as follows.

Definition 4.1. We say that a lifted single agent action α can be safely applied w.r.t. a set of preconditions $pre(\alpha)$ and CNFs $\{CNF_\ell\}_\ell$ if for every pb-literal $\langle \ell, b_{\ell,\alpha} \rangle$, either (1) the unit clause $\text{IsEff}(\langle \ell, b_{\ell,\alpha} \rangle, \alpha) \in CNF_\ell$, (2) the clause $\neg\text{IsEff}(\langle \ell, b_{\ell,\alpha} \rangle, \alpha) \in CNF_\ell$, or (3) $\langle \ell, b_{\ell,\alpha} \rangle \in pre(\alpha)$.

Finally, MA-SAM sets the effects of every action $\alpha \in A_{safe}$ to be the conjunction of all the pb-literals $\langle \ell, b_{\ell,\alpha} \rangle$ for which there exists a unit clause $\text{IsEff}(\langle \ell, b_{\ell,\alpha} \rangle, \alpha)$ in CNF_ℓ. Actions that cannot be safely applied, i.e., actions not in A_{safe}, are removed from the action model returned by MA-SAM, as we cannot accurately discern their

[4] Practically, we initialize the set of preconditions of an action by the set of pb-literals relevant to that action in one of the states in which it was observed.

Algorithm 1 MA-SAM Algorithm

1: **Input:** trajectories $\mathcal{T}$, lifted action names and parameters – $\{\alpha, params(\alpha) \in A\}$
2: **for** every lifted single-agent action α **do**
3: $pre(\alpha) \leftarrow$ all pb-literals that can be bound to α
4: $eff(\alpha) \leftarrow \emptyset$
5: **end for**
6: **for** every lifted literal ℓ, set $CNF_\ell = \emptyset$
7: **for** every $\langle s, \hat{a}, s' \rangle \in \mathcal{T}$ **do**
8: **for** every literal $l = \langle \ell, b_\ell \rangle$ not in s **do**
9: **for** every action $a = \langle \alpha, b_\alpha \rangle$ in $relevant(l, \hat{a})$ **do**
10: Remove $\left\langle \ell, b_{\ell,\alpha} \right\rangle$ from $pre(\alpha)$ ▷ Rule 1
11: **end for**
12: **if** $l \in s'$ **then**
13: $RelevantActions \leftarrow \{\alpha | a = \langle \alpha, b_\alpha \rangle \ \ where\ a \in relevant(l, \hat{a})\}$
14: Add $\bigvee_{\alpha \in RelevantActions}$ IsEff$(\left\langle \ell, b_{\ell,\alpha} \right\rangle, \alpha)$ to CNF_ℓ ▷ Rule 3
15: **end if**
16: **end for**
17: **for** every literal $l = \langle \ell, b_\ell \rangle$ not in s' **do**
18: **for** every action $a = \langle \alpha, b_\alpha \rangle$ in $relevant(l, \hat{a})$ **do**
19: Add $\neg$IsEff$(\left\langle \ell, b_{\ell,\alpha} \right\rangle, \alpha)$ to CNF_ℓ ▷ Rule 2
20: **end for**
21: **end for**
22: **end for**
23: **for** every lifted literal ℓ, minimize CNF_ℓ
24: $A_{safe} \leftarrow$ every observed action
25: **for** $\alpha \in A_{safe}$ and lifted literal ℓ **do**
26: **for** every pb-literal $pb = \left\langle \ell, b_{\ell,\alpha} \right\rangle$ not in $pre(\alpha)$ **do** ▷ check if pb must be an effect of α
27: **if** the unit clause $\{\text{IsEff}(pb, \alpha)\} \in CNF_\ell$ **then**
28: Add pb to $eff(\alpha)$
29: **else if** the unit clause $\{\neg\text{IsEff}(pb, \alpha)\} \notin CNF_\ell$ **then**
30: Remove α from A_{safe}
31: **end if**
32: **end for**
33: **end for**
34: **return** A_{safe}, pre, eff, $\{CNF_\ell\}_\ell$

effects given the available trajectories. In our pseudo-code (Alg. 1), we interleave the detection of safe-to-apply actions and the learning of their effects (lines 25-33).

Note that it is easy to obtain an action model M_i for each agent i from the action model returned by MA-SAM. Each agent is associated with a set of lifted actions whose groundings the agent was observed to perform. For every such lifted action, we add it to M_i along with the preconditions and effects found for it by MA-SAM if it is in A_{safe}.

Example 4.2 illustrates MA-SAM's learning process and how it resolves ambiguities between actions to infer a safe multi-agent action model.

Example 4.2. Consider an action model with the actions $(a_1\ ?x\ ?y\ ?z)$ and $(a_2\ ?w\ ?q)$. Assume that the domain contains only one fluent, $(\ell\ ?p_1\ ?p_2)$, and that we have observed two action triplets:

$$T_1 = \langle s_0, [(a_1\ o_1\ o_3\ o_2), (a_2\ o_1\ o_2)], s_1\rangle$$
$$T_2 = \langle s_2, [(a_1\ o_1\ o_3\ o_2), \textit{NO-OP}], s_3\rangle$$

Note that the joint action triplets T_1 and T_2 are not consecutive, i.e., s_1 is not the same as s_2.[5] The literal $\langle \ell\ o_1\ o_2\rangle$ appears in s_1 but not in s_0, s_2, or s_3. Now assume we run MA-SAM on these two action triplets. After processing the first action triplet (T1) MA-SAM will remove $\langle \ell\ ?x\ ?z\rangle$ and $\langle \ell\ ?w\ ?q\rangle$ from being considered as preconditions of a_1 and a_2, respectively (Rule 1). Subsequently, since applying the joint action resulted in the literal $\langle \ell\ o_1\ o_2\rangle$ being in $s_1 \setminus s_0$, MA-SAM will include the non-unit clause (Rule 3) in CNF_ℓ:

$$C_1 = \text{IsEff}(\langle \ell\ ?x\ ?z\rangle, a_1) \vee \text{IsEff}(\langle \ell\ ?w\ ?q\rangle, a_2).$$

After processing the second action triplet (T2) and observing that $\langle \ell\ o_1\ o_2\rangle$ is not in s_3, MA-SAM adds to CNF_ℓ the unit clause (Rule 2)

$$C_2 = \neg\text{IsEff}(\langle \ell\ ?x\ ?z\rangle, a_1),$$

and correspondingly prunes the clause C_1 using C_2 by applying unit propagation (line 33). As a result, MA-SAM can infer that $\langle \ell\ ?w\ ?q\rangle$ must be an effect of a_2, and will return $A_{safe} = \{a_1, a_2\}$ with a safe action model as follows:

$$\textit{pre}(a_1) = \emptyset, \textit{eff}(a_1) = \emptyset$$
$$\textit{pre}(a_2) = \emptyset, \textit{eff}(a_2) = \langle \ell\ ?w\ ?q\rangle$$

Note that if MA-SAM was given only T1, it would remove both actions from A_{safe}, as their effects would be ambiguous.

4.1 Theoretical Analysis

This section presents several theoretical properties of the MA-SAM algorithm.

4.1.1 Safety.

Lemma 4.3. *Let M and A_{safe} be the action model and set of actions returned by MA-SAM, respectively. The following properties hold for each action $\alpha \in A_{safe}$:*

(1) $\textit{eff}_M(\alpha) \subseteq \textit{eff}_{M^*}(\alpha)$.
(2) if $\langle \ell, b_{\ell,\alpha}\rangle \in \textit{eff}_{M^*}(\alpha)$ then either $\langle \ell, b_{\ell,\alpha}\rangle \in \textit{eff}_M(\alpha)$ or $\langle \ell, b_{\ell,\alpha}\rangle \in \textit{pre}_M(\alpha)$.

Proof. To prove the first property, assume by contradiction that there exists a pb-literal $\langle \ell, b_{\ell,\alpha}\rangle \in \textit{eff}_M(\alpha)$ such that $\langle \ell, b_{\ell,\alpha}\rangle \notin \textit{eff}_{M^*}(\alpha)$. Each pb-literal added to $\text{IsEff}(\langle \ell, b_{\ell,\alpha}\rangle)$ corresponds to a pb-literal observed as a result of a joint action $\hat{a}$ that includes a grounded execution of α. Since $\langle \ell, b_{\ell,\alpha}\rangle \notin \textit{eff}_{M^*}(\alpha)$, there must exist an action $\alpha' \neq \alpha$ such that $\langle \ell, b_{\ell,\alpha'}\rangle \in \textit{eff}_{M^*}(\alpha')$.

Given that $\hat{a}$ causes $\text{IsEff}(\langle \ell, b_{\ell,\alpha}\rangle)$ to be added to CNF_ℓ, both α and α' have been observed concurrently as part of $\hat{a}$. Consequently, since the effect results from applying either α or α', the clause $C = \text{IsEff}(\langle \ell, b_{\ell,\alpha}\rangle) \vee \text{IsEff}(\langle \ell, b_{\ell,\alpha'}\rangle)$ is added to CNF_ℓ. However, the clause $\neg\text{IsEff}(\langle \ell, b_{\ell,\alpha'}\rangle)$ will never be added to CNF_ℓ. Therefore, C remains a non-unit clause, leading MA-SAM to remove α from A_{safe}. This contradicts the initial assumption that $\alpha \in A_{safe}$; thus, $\langle \ell, b_{\ell,\alpha}\rangle \in \textit{eff}_{M^*}(\alpha)$.

To prove the second property, assume that $\langle \ell, b_{\ell,\alpha}\rangle \in \textit{eff}_{M^*}(\alpha)$. For that for every joint action triplet $\langle s, \hat{a}, s'\rangle$ involving a grounded execution of α, it must hold that either $l = \langle \ell, b_\ell\rangle$ belongs to s or s'. If $l \in s$, then according

[5] In general, MA-SAM's inductive rules (Observation 2) do not require the joint action triplets to be consecutive, as is the case in SAM.

Rule 1, $\langle \ell, b_{\ell,\alpha} \rangle \in pre_M(\alpha)$. Otherwise, there exists a CNF clause containing the term IsEff($\langle \ell, b_{\ell,\alpha} \rangle$). Since $\alpha \in A_{safe}$, either $\langle \ell, b_{\ell,\alpha} \rangle \in pre_M(\alpha)$ or IsEff($\langle \ell, b_{\ell,\alpha} \rangle$) must be a unit clause, implying $\langle \ell, b_{\ell,\alpha} \rangle \in eff_M(\alpha)$. □

Next, we use Lemma 4.3 to prove the action models learned by MA-SAM are safe.

Theorem 4.4. *The multi-agent action model M learned using MA-SAM is safe w.r.t. the real action model (M^*).*

Proof. For each action α in the multi-agent action model M learned by MA-SAM, it holds that $pre_M(\alpha)$ is a superset of the action's preconditions in M^*. Thus, for any state s and grounded action $a = \langle \alpha, b_\alpha \rangle$ such that $app_M(a, s)$ then a is also applicable according to M^*. The effects $eff_M(\alpha)$ are a subset of the effects in M^*, and each effect in $eff_{M^*}(\alpha)$ is either an effect of α according to M or is a precondition of the action. Thus, according to Observation 1, the action model learned using MA-SAM is safe w.r.t to the real agents' action models. □

4.1.2 Runtime.

Theorem 4.5. *Let k, N_{pb}, $N_{\mathcal{T}}$, and N_A, be the number of agents, pb-literals, action triplets, and single-agent actions, respectively, in the domain. The runtime of MA-SAM is $O(N_A \cdot N_{pb}^2 \cdot k \cdot N_{\mathcal{T}})$, i.e., linear in the number of agents, lifted actions, and action triplets, and quadratic in the number of pb-literals.*

Proof. The runtime complexity of applying MA-SAM's inductive rules on a given action triplet is linear in the number of agents and pb-literals. MA-SAM starts by applying these inductive rules on every action triplet in $\mathcal{T}$ and populating its CNFs. This requires runtime of $O(k \cdot N_{pb} \cdot N_{\mathcal{T}})$ Then MA-SAM minimizes the resulting CNFs by propagating unit clauses. The number of clauses in each CNF is at most linear in the number of agents, pb-literals for the corresponding literal, and action triplets. Since applying unit propagation on a CNF is linear in the total formula size, and there is a CNF for every literal, the complexity of minimizing our CNFs is $O(k \cdot N_{pb}^2 \cdot N_{\mathcal{T}})$. Finally, identifying the actions that can be safely applied and computing the effects of the safe actions requires runtime that is linear in the number of lifted actions, pb-literals, and CNF clauses, i.e., $O(N_A \cdot N_{pb}^2 \cdot k \cdot N_{\mathcal{T}})$. Thus, the complexity of MA-SAM is $O(N_A \cdot N_{pb}^2 \cdot k \cdot N_{\mathcal{T}})$, which is only linear in the number of agents, lifted actions, and action triplets, and quadratic in the number of pb-literals. □

4.1.3 Sample Complexity.

In this work, we are interested in learning an action model that can be used to solve problems in the real domain. Therefore, following prior work on learning safe action models (Juba, Le, et al. 2021; Juba and Stern 2022; Stern and Juba 2017), we consider a notion of sample complexity that is related to the problem-solving ability of the learned action model. To properly define this notion of sample complexity, we introduce the following notations and definitions.

Definition 4.6 (Enables solving). We say that an action model M enables solving an MA-PDDL problem $\Pi = \langle O, I, G \rangle$ if there exists a plan $\pi = (a_1, \ldots, a_n)$ that solves Π according to the preconditions and effects specified in M.

This means that if M enables solving Π then, given enough computing resources, a complete planner using M will be able to find a plan for Π.

Let D be a probability distribution over solvable MA-PDDL problems in a given domain. For a MA-PDDL problem Π, let $T(\Pi)$ be a probability distribution over trajectories corresponding to plans that solve Π. [6]

Definition 4.7 (Sample complexity). The sample complexity of planning with a learned safe action model for a given $\epsilon, \delta \in (0, 1)$ is defined as the minimum number of trajectories $m(\epsilon, \delta)$ such there exists an algorithm X that returns safe action models, such that for any distributions D and $T(\cdot)$, with probability at least $1 - \delta$ the safe action model returned by X on $m(\epsilon, \delta)$ trajectories enables solving a problem Π drawn from D with probability at least $1 - \epsilon$.

[6] $T(\Pi)$ may be produced, for example, by running a sound and complete planner on Π.

Based on Theorem 5 by (Juba, Le, et al. 2021), the sample complexity of learning a safe action model for single-agent PDDL problems is at most linear in the number of actions and state variables, and this is attained by the SAM algorithm. In our context, the number of joint actions is exponential in the number of agents, yielding an upper bound on the sample complexity that is exponential in the number of agents as well. We prove this formally below.

THEOREM 4.8. *For all integers $N_A \geq 2$, κ, k, and $N_F \geq 2N_Ak$, and real numbers $0 < \epsilon, \delta < 1/4$ there is a family of domains with k agents, each with N_A actions, and N_F propositional fluents, with associated distributions $T(\cdot)$ that assign nonzero probability only to trajectories in which at most κ agents participate in each joint action (i.e., the other $k - \kappa$ agents take NO-OP), such that the sample complexity for ϵ and δ is at least $\Omega\left(\frac{1}{\epsilon}(N_F N_A^{\kappa}\binom{k}{\kappa} + \log\frac{1}{\delta})\right)$.*

PROOF. The domains for Theorem 4.8 are as follows. Let $a_{i,j}$ denote the j^{th} action of agent i. For each agent i, the domain includes a set of N_A fluents $(f_{i,1}, \ldots, f_{i,N_A})$. Each fluent $f_{i,j}$ is an effect of the action $a_{i,j}$, and this is the only action with that fluent as an effect. We refer to these fluents as the agent-action fluents. The domain also includes an additional set of $N_F - N_Ak$ fluents, referred to as *flags*. All of the actions have all of the flags as effects. The actions have no other effects.

Now consider the following distribution on problems (D). The initial state of every problem sampled from D has all of the N_Ak agent-action fluents set to false false, and all but one of the flags (uniformly at random) true. With probability $1 - 4\epsilon$, the goal is empty; otherwise, the goal G is set as follows. First, we choose κ agents uniformly at random. Then, for each of the chosen agents we choose a single agent-action fluent uniformly at random from the agent-action fluents corresponding to actions of that agent. We set G to have (1) the chosen κ agent-action fluents true, (2) all other agent-action fluents false, and (3) all the flags true.

For this distribution of problems, we consider the following distribution over trajectories $T(\Pi)$. If Π is a problem where the goal is empty, $T(\Pi)$ returns a single trajectory where all agents take *NO-OP* actions. Otherwise, $T(\Pi)$ returns a trajectory with a single joint action in which the set of κ agents with actions corresponding to the agent-action fluents in the goal take the relevant actions to achieve the goal.

To obtain a success rate of $1 - \epsilon$ on problems drawn from D, we require an action model that enables solving at least $3/4$ of the problems with non-empty goals. This is because such problems occur with probability 4ϵ, and if we fail to solve more than $\frac{1}{4}$ of them we exceed a failure rate of $\frac{1}{4}4\epsilon = \epsilon$. Next, we compute the probability of learning such an action model by an algorithm that is guaranteed to return a safe action model. Let $\Pi = \langle F, k, \{A_i\}_{i-1}^{k}, I, G\rangle$ be a problem drawn from D where G is not empty. We denote by $f(I)$, $f(\kappa, G)$, and $actions(\kappa, G)$, the flag fluent that is false in I, the set of κ agent-action fluents that must be achieved in G, and the set of actions that achieves them, respectively. Clearly, any plan that solves Π must include the actions in $actions(\kappa, G)$ (concurrently or not) and no other action, as that would set additional agent-action fluents to be true, while they must be false for G to be satisfied. Consider the case where the set of trajectories given to the action model learning algorithm ($\mathcal{T}$) does not include a trajectory that starts in I and includes a joint action with exactly $actions(\kappa, G)$ (and all other agents performing *NO-OP*). In this case, the action model in which $f(I)$ is not an effect of any of the actions in $actions(\kappa, G)$, the actions in $actions(\kappa, G)$ have all other flags as effects, and all other actions have all flags (including those in $f(I)$) as effects is consistent with the given trajectory. Indeed, in any trajectory with a subset of $actions(\kappa, G)$, there is at least one other action that has $f(I)$ as an effect, so all of the flags are effects of the joint actions in these trajectories. Since such an action model exists, any safe action model cannot assume that $f(I)$ is an effect of the actions $actions(\kappa, G)$. Thus, Π cannot be solved in this case.

In summary, nonempty goals cannot be achieved unless we have previously observed an example trajectory in which the joint action consists of the κ relevant agent-action pairs, with the relevant flag false in the pre-state. There are at least $N_F/2$ flags (since $N_F \geq 2N_Ak$) and $N_A{}^{\kappa}\binom{k}{\kappa}$ such joint actions, which we observe uniformly at random among the non-empty goals. The probability that we fail to observe such an example among $\frac{1}{8}N_F N_A{}^{\kappa}\binom{k}{\kappa}$

of the trajectories with non-empty goals is at least

$$\left(1 - \frac{2}{N_F {N_A}^{\kappa} \binom{k}{\kappa}}\right)^{\frac{1}{8} N_F {N_A}^{\kappa} \binom{k}{\kappa}} > 1/2$$

since $1 - x + x^2 > e^{-x}$ for positive x and $k, \kappa \geq 1$, $N_A \geq 2$, and $N_F \geq 4$. The expected number of such goals, in turn, is therefore at least $1/2$ of them by the linearity of expectation; the probability that more than $3/4$ are observed is less than $2/3$ by Markov's inequality. In turn, observing that the number of examples with non-empty goals out of $\frac{1}{16\epsilon} N_F {N_A}^{\kappa} \binom{k}{\kappa}$ exceeds $\frac{1}{8} N_F {N_A}^{\kappa} \binom{k}{\kappa}$ with probability at most $1/2$ by Markov's inequality, we see that if the training set includes fewer than $\frac{1}{16\epsilon} N_F {N_A}^{\kappa} \binom{k}{\kappa}$ examples, with probability greater than $1/2 \cdot 2/3 > \delta$ over the training set, any safe action model for the training set must fail to achieve the goal with probability greater than ϵ.

Furthermore, we observe that a training set of at least $\frac{1}{\epsilon} \ln \frac{1}{\delta}$ examples are also necessary, since if we draw fewer than this many examples, the probability that we never observe an example with a nonempty goal is at least $(1 - 8\epsilon)^{\frac{1}{\epsilon} \ln \frac{1}{\delta}} > e^{-\ln \frac{1}{\delta}} = \delta$. In such a case, no safe action model can take any actions, thus suffering a failure rate of at least 8ϵ. As $(x + y)/2 \leq \max(x, y)$, we can therefore conclude that more than $\frac{1}{32\epsilon}(N_F {N_A}^{\kappa} \binom{k}{\kappa} + \ln \frac{1}{\delta})$ examples are necessary. □

Theorem 4.8 provides a *worst case* analysis of the sample complexity required to learn an MA-PDDL action model. Of course, it may be easier to learn MA-PDDL models given some assumptions about the distributions of trajectories. We remark that the special case $k = 1$ corresponds to single-agent PDDL domains. For this special case, Theorem 4.8 gives a lower bound of

$$\Omega\left(\frac{1}{\epsilon}(N_F N_A + \log \frac{1}{\delta})\right)$$

on the sample complexity. This matches the upper bound attained by SAM learning (Juba, Le, et al. 2021, Theorem 5). Therefore, Theorem 4.8 also proves that SAM learning has an optimal sample complexity, settling the asymptotic sample complexity of learning safe action models for single-agent PDDL environments.

Unfortunately, the number of trajectories required to provide such an ϵ-δ guarantee for MA-SAM can be unbounded. To illustrate, consider the following example:

Example 4.9. In this example, the actions $(a_1\ ?x\ ?y\ ?z)$ and $(a_2\ ?w\ ?q)$ are always observed being executed concurrently as follows:

$$\langle s_i, [(a_1\ o_1\ o_3\ o_2), (a_2\ o_1\ o_2)], s_{i+1} \rangle$$

In addition, the literal $\langle \ell, o_1, o_2 \rangle$ appears in s_{i+1} but not in s_i. Thus, it must be an effect of either a_1, a_2, or both. Running MA-SAM in this case will result in CNF_ℓ having a non-unit clause $C = \text{IsEff}(\langle \ell\ ?x\ ?z \rangle, a_1) \vee \text{IsEff}(\langle \ell\ ?w\ ?q \rangle, a_2)$. Consequently, it is impossible to disambiguate between the effects of a_1 and a_2, and neither will be included in A_{safe}. In such a case, no number of trajectories will allow MA-SAM to safely apply these actions, even concurrently. This can severely limit the number of problems that can be solved with the learned action model.

5 MA-SAM with Macro Actions

An alternative to MA-SAM that does not suffer from the limitation above is to use the (single-agent) SAM algorithm to directly learn an action model for *every joint action* instead of using MA-SAM. We call this algorithm *SAM over Joint Actions (SAMoJA)*. Given a planning domain with k agents, SAMoJA attempts to learn a single action model that explicitly defines the preconditions and effects of every possible joint action. This is different from MA-SAM, which learns a set of single-agent action models, which can then be used to compose joint actions

(by the MA-PDDL planner). Consider running SAMoJA on the scenario specified in Example 4.9, assuming only two agents exist, where the first agent can perform action a_1 and the second can perform action a_2. SAMoJA will return an action model that defines separate preconditions and effects for every possible joint action:

$$[a_1, \textit{NO-OP}], [\textit{NO-OP}, a_2], [a_1, a_2].$$

SAMoJA avoids the need for ambiguity resolution since it simply learns the preconditions and effects of each joint action directly, treating it as a single action.

Based on the properties of SAM, we know that SAMoJA is guaranteed to eventually learn a safe (joint) action model that, with high probability, will enable solving most problems in the domain (Juba, Le, et al. 2021).

However, SAMoJA is not a practical algorithm for learning a multi-agent action model for the following reasons:

(1) **SAMoJA does not output an action model for each agent.** Instead, it outputs an action model for a single, multi-armed, agent. Thus, one cannot use a multi-agent planning algorithm, such as MAFS (Nissim and R. Brafman 2014), GPPP (Maliah et al. 2016a), and others, to plan with the output of SAMoJA. Instead, a single-agent planner must be used to plan with the output of SAMoJA.
(2) **SAMoJA outputs an exponentially large action model.** The number of joint actions grows exponentially with the number of agents. Therefore, the action model returned by SAMoJA can be too large to be useful for off-the-shelf single-agent planners.
(3) **SAMoJA cannot generalize beyond previously observed joint actions.** SAMoJA can only learn the preconditions and effects of joint actions that have been explicitly seen during training. For example, in the setting described in Example 4.2, unless SAMoJA observes a transition of the form $\langle s_m, [\textit{NO-OP}, a_2], s_{m+1}\rangle$, its learned action model will not allow the execution of the joint action $[\textit{NO-OP}, a_2]$. In contrast, MA-SAM learns the individual agent's actions independently, enabling it to generalize across different joint executions. As a result A_{safe} and the corresponding action model produced by MA-SAM include both a_1 and a_2.

Next, we propose *MA-SAM with Macro Actions* (MA-SAM$^+$), an algorithm that attempts to learn individual agents' action models whenever possible, and resorts to learning joint executions only when necessary. Specifically, MA-SAM$^+$ optimistically applies MA-SAM to learn the actions of single agents. When ambiguities cannot be resolved through learning, it backtracks and applies an approach similar to SAMoJA over the remaining ambiguous actions, effectively learning a model of their joint execution.

5.1 The MA-SAM$^+$ Algorithm

To explain the MA-SAM$^+$ algorithm, we first introduce the notion of a Lifted Macro Action (LMA). A LMA represents the concurrent execution of a set of single-agent actions.[7] Formally, a LMA $\mathcal{A}$ is a tuple $\langle \textit{actions}(\mathcal{A}), \textit{params}(\mathcal{A}), b_{\mathcal{A}}\rangle$ where $\textit{actions}(\mathcal{A})$ is a set of lifted single-agent actions, $\textit{params}(\mathcal{A})$ are the parameters of the LMA, and $b_{\mathcal{A}}$ is a binding that maps each parameter of the single-agent actions in $\textit{actions}(\mathcal{A})$ to a parameter in $\textit{params}(\mathcal{A})$. The MA-SAM$^+$ algorithm augments the action model returned by MA-SAM by adding LMAs for sets of single-agent actions, where each LMA includes at least one single-agent action that cannot be safely applied. In Example 4.9, where a_1 and a_2 are always observed concurrently, MA-SAM$^+$ may create the following LMA:

$$\mathcal{A}_{a_1,a_2} = \langle \{a_1, a_2\}, (?p1, ?p2, ?p3), \{?x, ?w \rightarrow ?p1, ?y \rightarrow ?p2, ?z, ?q \rightarrow ?p3\}\rangle$$

MA-SAM$^+$ learns preconditions and effects of each LMA such that if the preconditions of a LMA $\mathcal{A}$ hold, then all the single-agent actions in $\textit{actions}(\mathcal{A})$ must be applicable and the results of applying them concurrently are

[7] This is different from the traditional notion of macro actions, which is a *sequence* of actions (Botea et al. 2005; Hofmann et al. 2020; Korf 1985). In contrast, a LMA is a set of single-agent actions executed concurrently.

not ambiguous. We discuss later how MA-SAM$^+$ computes the preconditions and effects of a given LMA. Before introducing the details of MA-SAM$^+$, we provide the following definition.

Definition 5.1 (Consistent LMA). Let C be a clause in CNF_ℓ returned by MA-SAM and $actions(C)$ be the set of actions mentioned in a clause C, i.e., $actions(C) = \{\alpha |\ \text{IsEff}(\langle \ell, b_{\ell,\alpha} \rangle, \alpha) \in C\}$. We say that a LMA $\mathcal{A} = \langle actions(\mathcal{A}), params(\mathcal{A}), b_{\mathcal{A}} \rangle$ is *consistent* with C if it satisfies the following conditions:

(1) **Action Validity.** For every atom $\text{IsEff}(\langle \ell, b_{\ell,\alpha} \rangle, \alpha) \in C$, the action α is in $actions(\mathcal{A})$, i.e., $actions(C) \subseteq actions(\mathcal{A})$.
(2) **Parameter Agreement.** For every pair of single-agent actions $\alpha, \alpha' \in actions(\mathcal{A})$, every pair of atoms $\text{IsEff}(\langle \ell, b_{\ell,\alpha} \rangle, \alpha)$ and $\text{IsEff}(\langle \ell, b_{\ell,\alpha'} \rangle, \alpha')$ in C, and every parameter $p \in params(\ell)$, it holds that $b_{\mathcal{A}}(b_\alpha(p)) = b_{\mathcal{A}}(b_{\alpha'}(p))$.

In words, $\mathcal{A}$ is consistent with C if it includes all actions in $actions(C)$, and for every pair of single-agent actions $\alpha, \alpha' \in actions(\mathcal{A})$, the parameters used in $\langle \ell, b_{\ell,\alpha} \rangle$ and $\langle \ell, b_{\ell,\alpha'} \rangle$ are bound to the same parameter in $\mathcal{A}$. This means applying $\mathcal{A}$ is guaranteed to result in the effect specified in C.

Algorithm 2 MA-SAM$^+$

1: $A_{safe}, pre, eff, \{CNF_\ell\}_\ell \leftarrow$ MA-SAM
2: $\mathcal{A}s \leftarrow$ FindCandidateLMAs($A_{safe}, \{CNF_\ell\}_\ell$)
3: **for** every $\mathcal{A} = \langle actions(\mathcal{A}), params(\mathcal{A}), b_{\mathcal{A}} \rangle$ in $\mathcal{A}s$ **do**
4: $pre(\mathcal{A}) \leftarrow \bigcup_{a \in actions(\mathcal{A})} pre(a)$; $eff(\mathcal{A}) \leftarrow \emptyset$
5: **for** every lifted literal ℓ and clause $C \in CNF_\ell$ **do**
6: **if** $actions(C) \cap actions(\mathcal{A}) = \emptyset$ **then**
7: continue to the next clause
8: **else**
9: **if** C is consistent with $\mathcal{A}$ **then** ▷ ℓ is an effect of $\mathcal{A}$
10: Add $\bigcup_{\text{IsEff}(\langle \ell, b_{\ell,\alpha} \rangle, \alpha) \in C} \langle \ell, b_{\ell,\alpha} \rangle$ to $eff(\mathcal{A})$
11: **else** ▷ ℓ may or may not be an effect of $\mathcal{A}$. Add it to pre to ensure safety.
12: **for** $\text{IsEff}(\langle \ell, b_{\ell,\alpha} \rangle, \alpha) \in C$ s.t. $\alpha \in actions(\mathcal{A})$ **do**
13: Add $\langle \ell, b_{\ell,\alpha} \rangle$ to $pre(\mathcal{A})$
14: **end for**
15: **end if**
16: **end if**
17: **end for**
18: Add $\mathcal{A}$ to A_{safe}; $pre(\mathcal{A})$ to pre; and $eff(\mathcal{A})$ to eff.
19: **end for**
20: **return** A_{safe}, pre, eff

Algorithm 2 presents the pseudo-code for MA-SAM$^+$. MA-SAM$^+$ starts by running MA-SAM, which returns (1) a set of single-agent actions A_{safe} that can be safely applied, (2) a set of preconditions and effects ($pre(\alpha)$ and $eff(\alpha)$) for every single-agent action α, and (3) a CNF_ℓ for every lifted literal ℓ. MA-SAM$^+$ then generates a set of LMAs by calling Algorithm 3, which we describe later. Next, MA-SAM$^+$ computes the preconditions and effects of each LMA $\mathcal{A} = \langle actions(\mathcal{A}), params(\mathcal{A}), b_{\mathcal{A}} \rangle$. The preconditions of $\mathcal{A}$ are initialized as the union of the preconditions of all its constituent single-agent actions according to pre (line 4). For the effects, MA-SAM$^+$ initializes $eff(\mathcal{A})$ as an empty set (line 4). Then, it iterates over every lifted literal ℓ and every clause $C \in CNF_\ell$. If C does not involve any actions in $actions(\mathcal{A})$, i.e., $actions(C) \cap actions(\mathcal{A}) = \emptyset$, MA-SAM$^+$ skips to the next clause. Otherwise, it checks whether C is consistent with $\mathcal{A}$, as defined in Definition 5.1. If C is consistent with

Algorithm 3 Lifted Macro Actions Candidate Generator

1: **Function: FindCandidateLMAs**(A_{safe}, $\{CNF_\ell\}_\ell$)
2: $Candidates \leftarrow \emptyset$
3: **for** every lifted literal ℓ and clause $C \in CNF_\ell$ **do**
4: **if** $\exists \alpha \in actions(C)$ s.t. $\alpha \notin A_{safe}$ **then**
5: Add $actions(C)$ to $Candidates$
6: **end if**
7: **end for**
8: $Candidates \leftarrow$ All unions of elements in $Candidates$
9: $\mathcal{A}s \leftarrow \emptyset$
10: **for** $A_{ca} \in Candidates$ **do**
11: $\mathcal{A} \leftarrow$ a new LMA with $actions(\mathcal{A}) \leftarrow A_{ca}$
12: $G \leftarrow$ CreateBindingGraph($\mathcal{A}$, $\{CNF_\ell\}_\ell$))
13: $b_{\mathcal{A}} \leftarrow \emptyset$
14: **for** every connected component $Comp$ in G **do** ▷ Bind action parameters
15: Create new LMA parameter $p_{\mathcal{A}}$ and add to $params(\mathcal{A})$
16: **for** $\langle \alpha, p_\alpha \rangle \in Comp$ **do**
17: Add $b_{\mathcal{A}}(\langle \alpha, p_\alpha \rangle) \leftarrow p_{\mathcal{A}}$
18: **end for**
19: **end for**
20: **for** every action $\alpha \in actions(\mathcal{A})$, $p_\alpha \in params(\alpha)$ **do** ▷ Handling unbound action parameters
21: **if** $\langle \alpha, p_\alpha \rangle$ not in $b_{\mathcal{A}}$ **then**
22: Add a new parameter p' to $params(\mathcal{A})$
23: $b_{\mathcal{A}}(\langle \alpha, p_\alpha \rangle) \leftarrow p'$
24: **end if**
25: **end for**
26: Add $\mathcal{A}$ to $\mathcal{A}s$
27: **end for**
28: **return** $\mathcal{A}s$

$\mathcal{A}$, MA-SAM$^+$ adds the effect specified in C, with the appropriate parameter binding from $b_{\mathcal{A}}$, to $eff(\mathcal{A})$ (line 10). Otherwise, C is not consistent with $\mathcal{A}$. This means $\mathcal{A}$ may or may not achieve the pb-literal represented by the clause C. Consequently, MA-SAM$^+$ adds the corresponding pb-literal to $pre(\mathcal{A})$ to ensure safety (line 13). Finally, MA-SAM$^+$ adds the LMA to A_{safe} and updates pre and eff accordingly.

Algorithm 3 explains the LMA candidate generation process. MA-SAM$^+$ initializes $Candidates$ as an empty set. Then, MA-SAM$^+$ iterates over every lifted literal ℓ and clause C in CNF_ℓ, and checks if there exists an atom $\text{IsEff}(\langle \ell, b_{\ell,\alpha} \rangle, \alpha) \in C$ that includes an unsafe action, i.e., where $\alpha \notin A_{safe}$. If such an atom exists, MA-SAM$^+$ adds the set $actions(C)$ to $Candidates$ (lines 3-7). Once all lifted literals and clauses have been processed, MA-SAM$^+$ adds every union of elements in $Candidates$ to $Candidates$ (line 8).

Next, MA-SAM$^+$ generates a LMA $\mathcal{A}$ for each set of actions $A_{ca} \in Candidates$ (line 11). Then it creates what we refer to as the *parameter binding graph*, denoted G. In this graph, a node represents a parameter of a single-agent action, i.e., a pair of the form $\langle \alpha, p \rangle$, where $p \in params(\alpha)$. There is an edge between two nodes $\langle \alpha, p_\alpha \rangle$ and $\langle \alpha', p_{\alpha'} \rangle$ in G if there exists a lifted literal ℓ, a parameter $p \in params(\ell)$, and a clause $C \in CNF_\ell$ such that both p_α and $p_{\alpha'}$ are bounded to p. That is, C includes an atom $\text{IsEff}(\langle \ell, b_{\ell,\alpha} \rangle, \alpha)$ and an atom $\text{IsEff}(\langle \ell, b_{\ell,\alpha'} \rangle, \alpha')$ such that $b_{\ell,\alpha}(p) = p_\alpha$ and $b_{\ell,\alpha'}(p) = p_{\alpha'}$. Figure 2 presents a visual illustration of the graph created for Example 4.9.

Every connected component in the binding graph represents a set of single-agent action parameters that have been bound to the same lifted literal parameter in some clause. Thus, we create a LMA parameter $p_{\mathcal{A}}$ for

every connected component in G, and add to $b_{\mathcal{A}}$ appropriate bindings from $p_{\mathcal{A}}$ to the single-agent actions' parameters in the corresponding connected component. We note that single-agent parameters not observed in any clause are added directly to the LMA (lines 20-25). Such parameters may exist due to the single-agent actions' preconditions.[8] In Example 4.9, we have two connected components: $\{\langle a_1, ?x\rangle, \langle a_2, ?w\rangle\}$ and $\{\langle a_1, ?z\rangle, \langle a_2, ?q\rangle\}$. The generated LMA has two parameters generated from the parameter binding graph. The first is bound to $\langle a_1, ?x\rangle$ and $\langle a_2, ?w\rangle$, and the second to $\langle a_1, ?z\rangle$ and $\langle a_2, ?q\rangle$. The LMA also has a third parameter, $?y$, which was added directly from the action a_1.

Example 5.2. Following our previous example, assume that there exists an additional action $(a_3\ ?m)$ with no preconditions and its effect is the pb-literal $(\ell'\ ?m)$. In addition, we are given additional joint action triplet in which a_1 and a_3 are executed concurrently as follows:

$$T = \langle s_n, [(a_1\ o_1\ o_2\ o_3), (a_3\ o1)], s_{n+1}\rangle$$

The CNFs resulting from running MA-SAM on the joint action triplets including T, include the following clauses: $\text{IsEff}(\langle \ell\ ?x\ ?z\rangle, a_1) \vee \text{IsEff}(\langle \ell\ ?w\ ?q\rangle, a_2)$, $\text{IsEff}(\langle \ell'\ ?x\rangle, a_1) \vee \text{IsEff}(\langle \ell'\ ?m\rangle, a_3)$, and $\text{IsEff}(\langle \ell\ ?x\ ?z\rangle, a_1))$. Consequently, MA-SAM$^+$ will create LMAs for three sets of actions: $\{a_1, a_2\}$, $\{a_1, a_3\}$, and $\{a_1, a_2, a_3\}$. The binding graph created for $\mathcal{A}_{a_1,a_2}$ is presented in Figure 2, and the binding graphs created for $\mathcal{A}_{a_1,a_2,a_3}$ and $\mathcal{A}_{a_1,a_3}$ are presented in Figure 3. The resulting LMAs are as follows:

$$\mathcal{A}_{a_1,a_2} = \langle\{a_1, a_2\}, (?p1, ?p2, ?p3), \{?x, ?w \rightarrow ?p1, ?y \rightarrow ?p2, ?z, ?q \rightarrow ?p3\}\rangle$$

$$\mathcal{A}_{a_1,a_3} = \langle\{a_1, a_3\}, (?p1, ?p2, ?p3), \{?x, ?m \rightarrow ?p1, ?y \rightarrow ?p2, ?z \rightarrow ?p3\}\rangle$$

$$\mathcal{A}_{a_1,a_2,a_3} = \langle\{a_1, a_2, a_3\}, (?p1, ?p2, ?p3), \{?x, ?w, ?m \rightarrow ?p1, ?y \rightarrow ?p2, ?z, ?q \rightarrow ?p3\}\rangle$$

The action $\mathcal{A}_{a_1,a_2}$ is consistent with the clauses $\text{IsEff}(\langle \ell\ ?x\ ?z\rangle, a_1) \vee \text{IsEff}(\langle \ell\ ?w\ ?q\rangle, a_2)$ and $\text{IsEff}(\langle \ell\ ?x\ ?z\rangle, a_1))$. Thus, MA-SAM$^+$ will add $\langle \ell\ ?p1\ ?p3\rangle$ to $\mathit{eff}(\mathcal{A}_{a_1,a_2})$. $\mathcal{A}_{a_1,a_2}$ is not consistent with the clause $\text{IsEff}(\langle \ell'\ ?x\rangle, a_1) \vee \text{IsEff}(\langle \ell'\ ?m\rangle, a_3)$, and MA-SAM$^+$ will add $\langle \ell'\ ?p1\rangle$ to $\mathit{pre}(\mathcal{A}_{a_1,a_2})$.

The action $\mathcal{A}_{a_1,a_3}$ is consistent with the clauses $\text{IsEff}(\langle \ell\ ?x\ ?z\rangle, a_1))$ and $\text{IsEff}(\langle \ell'\ ?x\rangle, a_1) \vee \text{IsEff}(\langle \ell'\ ?m\rangle, a_3)$. Consequently, MA-SAM$^+$ will add the pb-literals $\langle \ell\ ?p1\ ?p3\rangle$ and $\langle \ell'\ ?p1\rangle$ to $\mathit{eff}(\mathcal{A}_{a_1,a_3})$. $\mathcal{A}_{a_1,a_3}$ is not consistent with the clause $\text{IsEff}(\langle \ell\ ?x\ ?z\rangle, a_1) \vee \text{IsEff}(\langle \ell\ ?w\ ?q\rangle, a_2)$, thus, MA-SAM$^+$ will add $\langle \ell\ ?p1\ ?p3\rangle$ to $\mathit{pre}(\mathcal{A}_{a_1,a_3})$[9]. Lastly, the action $\mathcal{A}_{a_1,a_2,a_3}$ is consistent with all the three clauses, resulting in $\mathit{pre}(\mathcal{A}_{a_1,a_2,a_3})$ remaining as an empty set, and $\mathit{eff}(\mathcal{A}_{a_1,a_2,a_3}) = \{\langle \ell\ ?p1\ ?p3\rangle, \langle \ell'\ ?p1\rangle\}$.

5.2 Theoretical Analysis

In this section, we analyze the theoretical properties of the MA-SAM$^+$ algorithm.

5.2.1 Safety. Let M and $\mathcal{A}s$ be the action model and the LMAs returned by MA-SAM$^+$. We denote by $\mathcal{A}_M(s)$ the state obtained by applying $\mathcal{A}$ according to the learned action model M.

Lemma 5.3. *For every state s and every LMA $\mathcal{A} = \langle \mathit{actions}(\mathcal{A}), \mathit{params}(\mathcal{A}), b_{\mathcal{A}}\rangle \in \mathcal{A}s$, if the grounded action $\mathcal{A}_G = \left\langle \mathcal{A}, b_{\mathcal{A}_G}\right\rangle$ is applicable in a state s, then following properties hold: (1) every single-agent action $\langle \alpha, b_\alpha\rangle \in \mathit{actions}(\mathcal{A}_G)$ is also applicable in s and, (2) the state obtained by applying $\mathcal{A}_{G_M}(s)$ on s is exactly the same as the result of applying the constituent single-agent actions on s according to M^*.*

Since $\mathit{pre}(\mathcal{A})$ includes at least all the preconditions in $\mathit{pre}(\alpha)$ for each $\alpha \in \mathit{actions}(\mathcal{A})$, the first statement trivially holds. Next, we prove the second statement.

[8] Another reason for having such parameters is in cases where an action has a parameter of a specific type, whose purpose of this parameter is ensuring the action is only applicable in problems where an object of that type exists.

[9] Notice that in this case, the pb-literal $\langle \ell\ ?p1\ ?p3\rangle$ was added both as a precondition and as an effect of the action.

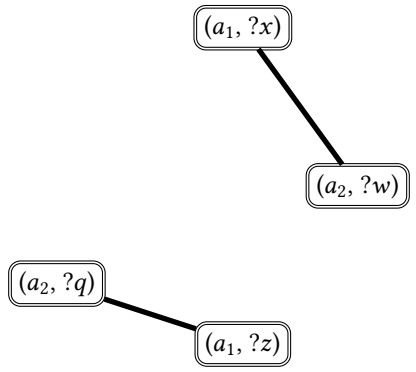

Fig. 2. The graphical illustration of the parameter binding graph created by MA-SAM$^+$ for the actions presented in Example 4.9.

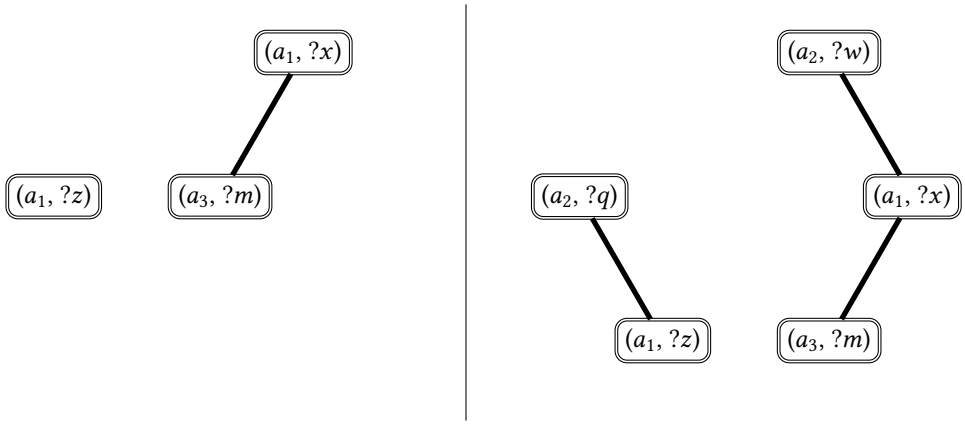

Fig. 3. The binding graphs for two LMAs in Example 5.2. The left figure represents the binding graph for the LMA $\mathcal{A}_{a_1,a_3}$ and the right graph represents the binding graph for the LMA $\mathcal{A}_{a_1,a_2,a_3}$.

Proof. For every LMA $\mathcal{A} \in \mathit{LMAs}$ and every pb-literal $\langle \ell, b_{\ell,\mathcal{A}} \rangle$ in the domain, one of the following holds:

(1) $\langle \ell, b_{\ell,\mathcal{A}} \rangle$ is a precondition of $\mathcal{A}$.
(2) $\langle \ell, b_{\ell,\mathcal{A}} \rangle$ is an effect of $\mathcal{A}$ and an effect of at least one of the corresponding single-agent actions according to M^*.
(3) $\langle \ell, b_{\ell,\mathcal{A}} \rangle$ is not an effect of $\mathcal{A}$ and it is not an effect of any of the corresponding single-agent actions according to M^*.

If $\langle \ell, b_{\ell,\mathcal{A}} \rangle \in \mathit{eff}(\mathcal{A})$, then there exists a CNF clause C with which $\mathcal{A}$ is consistent. Consequently, there is a joint action triplet in the given trajectories that resulted in adding the pb-literal to the next state. Thus, the execution of the actions in $\mathit{actions}(C)$ results in the same effect according to M^*.

Next, consider the case where $\langle \ell, b_{\ell,\mathcal{A}} \rangle$ is neither a precondition of $\mathcal{A}$ nor an effect of $\mathcal{A}$. Since $\langle \ell, b_{\ell,\mathcal{A}} \rangle$ is not a precondition of $\mathcal{A}$, then it is not a precondition of any of its constituent single-agent actions. This means that for every lifted action $\alpha \in \mathit{actions}(\mathcal{A})$ we observed a joint action triplet $\langle s, \hat{a}, s' \rangle$ that included $\langle \alpha, b_\alpha \rangle$ in which $\langle \ell, b_\ell \rangle$ was not true in s. Since $\langle \ell, b_{\ell,\mathcal{A}} \rangle$ is not an effect of $\mathcal{A}$, then $\langle \ell, b_\ell \rangle$ was also not true in s'. Otherwise, a CNF

clause C with the atom IsEff($\langle \ell, b_{\ell,\alpha} \rangle, \alpha$) which $\mathcal{A}$ is consistent with would exist, resulting in adding $\langle \ell, b_{\ell,\mathcal{A}} \rangle$ to $\mathit{eff}(\mathcal{A})$ contradicting our previous assumption. Thus, for every action in $\mathit{actions}(\mathcal{A})$, the pb-literal is not an effect according to M^*

□

Theorem 5.4. *The multi-agent action model M learned using MA-SAM$^+$ is safe w.r.t. the real action model (M^*).*

Proof. From Theorem 4.4, we know that the action model containing only the actions in A_{safe} is safe w.r.t. M^*. For every LMA $\mathcal{A} \in \mathcal{A}s$, the preconditions of $\mathcal{A}$ are a superset of the preconditions of the actions in $\mathit{actions}(\mathcal{A})$. Thus, for each state s, and grounded action $\mathcal{A}_G$, if $app_M(\mathcal{A}_G, s)$, then for any grounded action a in $\mathit{actions}(\mathcal{A}_G)$, it holds that $app_{M^*}(a, s)$. Furthermore, from Lemma 5.3, we know that applying $\mathcal{A}_G$ on a state results in the same state as if applying the actions in $\mathit{actions}(\mathcal{A}_G)$ according to M^*. Thus, the action model learned using MA-SAM$^+$ is safe to apply w.r.t. the real agents' action model. □

5.2.2 *Runtime.*

Theorem 5.5. *Let k, N_{pb}, $N_{\mathcal{T}}$, and N_A be the number of agents, pb-literals, action triplets, and single-agent actions, respectively, in the domain. The runtime of MA-SAM$^+$ is $O({N_A}^k \cdot k \cdot N_{pb}^2 \cdot N_{\mathcal{T}})$, i.e., exponential in the number of agents, quadratic in the number of pb-literals, and linear in the number action triplets.*

Proof. First, MA-SAM$^+$ applies MA-SAM, which has a runtime complexity of $O(k \cdot N_A \cdot N_{pb}^2 \cdot N_{\mathcal{T}})$. Next, it calls the FindCandidateLMAs algorithm. This algorithm starts by iterating over all CNFs, creating the set *Candidates*, which in the worst case includes all subsets of single-agent actions (${N_A}^k$). For each action, constructing the parameter binding graph is linear in the number of atoms in all CNFs, i.e., $O(k \cdot N_{pb}^2 \cdot N_{\mathcal{T}})$. Generating the new LMA parameters from the connected components is linear in the number of components. Adding the preconditions and the effects for each LMA is linear in the number of CNF clauses. Thus, the total runtime complexity is exponential in the number of agents, quadratic in the number of pb-literals, and linear in the number of action triplets. □

6 Experimental Results

We conducted a set of experiments to evaluate the action model learning algorithms presented in this paper, namely MA-SAM and MA-SAM$^+$. As discussed in Section 2.2, there is no natural baseline to compare with MA-SAM, as existing approaches to learning action models do not support learning from trajectories containing joint actions. Therefore, we used the SAM algorithm (Juba, Le, et al. 2021) as a baseline, having it ignore observed joint actions that include more than one single-agent action. We chose not to implement SAMoJA due to its inability to generalize beyond previously observed joint actions, and because its action models can become prohibitively large in domains with many agents.

For our experiments, we used domains from the publicly available Competition of Distributed and Multi-agent Planners (CoDMAP) benchmark (Štolba et al. 2016). This standard MA-STRIPS benchmark includes 12 MA-STRIPS domains with 20 problems each. We used seven of these domains, namely, Blocks, Depots, Driverlog, Logistics, Rovers, Satellite, and Sokoban. The remaining domains were excluded for the following reasons: the Taxi domain encoded passenger and taxi identities directly in the action names (e.g., $enter\_p1$, $exit\_p1$), preventing generalization; Woodworking and Elevators required action costs, which are unsupported; problems in Wireless were unsolvable using the real agents' models; and in Zenotravel, all plans involved a single agent executing actions sequentially, rendering the domain irrelevant for our setting. We note that in the Rovers domain, some actions may violate the injective binding assumption. If an algorithm encounters such a triplet, the triplet is discarded.

We also introduce a novel domain that is based on the popular video game Overcooked (S. A. Wu et al. 2021). In this domain, two agents cooperate to prepare meals for customers as efficiently as possible. The agents must

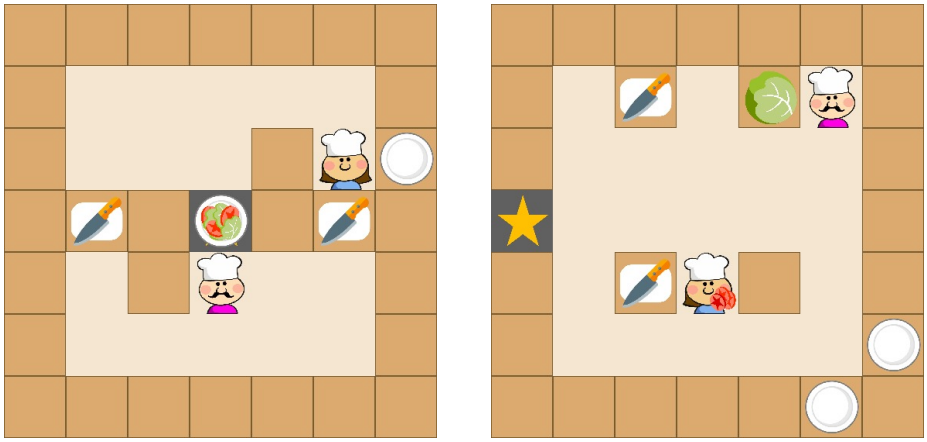

Fig. 4. A visual illustration of the task the agents executed in the Overcooked benchmark.

navigate in the kitchen to locations containing utensils and vegetables, chop the ingredients, and deliver the prepared meals to the customers.[10] Figure 4 illustrates example tasks from our benchmark, generated using the API provided by S. A. Wu et al. (2021).

In each experiment, we collected a set of planning problems in the same domain. This set was then split into training and test sets using an 80:20 ratio, and we used a planner to find plans for the problems in the training set. Then, we generated trajectories including joint action triplets from these plans and split them into action triplets (see Section 6.1 for more details). The joint action triplets are given to the evaluated algorithm, which outputs an MA-PDDL action model. We then evaluate the resulting action model by comparing it to the real action model of that domain, and by evaluating the ability of a planner to solve the test problems using this action model (see Section 6.2 for more details). To ensure robustness, we employed a 5-fold cross-validation strategy, repeating each experiment five times with different problem samples for training and testing. The implementation of the algorithms and the dataset used for the experiments are available in https://github.com/SPL-BGU/ma-sam. All experiments were conducted on a cluster with 8 cores and 32 GB of RAM.

6.1 Generating Trajectories of Joint Action Triplets

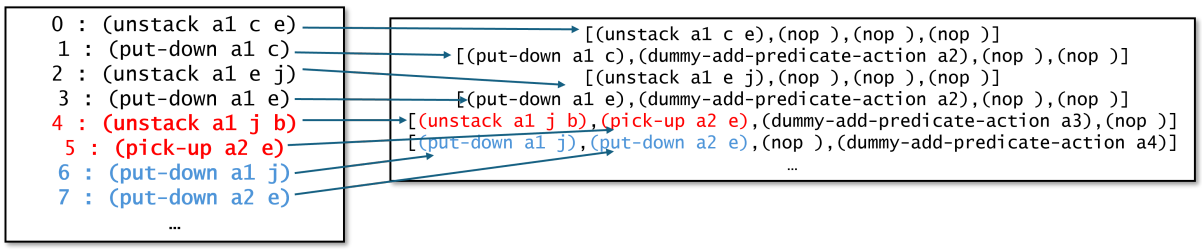

Fig. 5. Blocks domain's multi-agent sequential plan converted to a concurrent plan by combining well-defined actions, and the new dummy actions to the same joint actions. The actions merged into the same joint actions are marked in red and blue.

[10]The Overcooked domain, problems and trajectories are available as part of the experiment dataset.

To generate trajectories of joint action triplets for learning, we generated plans for the problems in the evaluated domain using two MA-PDDL solvers: GPPP (Maliah et al. 2017) with landmarks and FF heuristics and MAFS (Nissim and R. Brafman 2014) with the FF heuristic. Like most existing MA-STRIPS solvers, the above solvers output a plan π, which is a sequence of single-agent actions. One way to transform a trajectory of single-agent actions to a trajectory of joint actions is by creating for every single agent action a, a joint action in which one agent performs a and all other agents do nothing (*NO-OP*). We refer to such a joint action, in which only one agent acts, as a *trivial joint action*. For example, the joint action [*NO-OP*, *NO-OP*, $(a_2\ ?w\ ?q)$] is a trivial joint action.

Clearly, learning only from trajectories of trivial joint actions does not evaluate our algorithms' ability to learn from observations with concurrently executed actions. Therefore, we transformed every sequential plan created by the MA-PDDL planner to a plan with joint actions by grouping the maximal number of single-agent actions that form well-defined joint actions (Crosby, Jonsson, et al. 2014). Figure 5 illustrates the grouping of single-agent actions to joint actions. In the illustrated example, the single-agent actions highlighted in red are grouped to one joint action, and the single-agent actions highlighted in blue are grouped to a different joint action.

We observed empirically that the process described above for grouping single-agent actions in the sequential plan to joint actions still generated trajectories that included mostly trivial joint actions. This may be an artifact of the benchmark domains or the MA-PDDL planners, which do not aim for plans that can be parallelized. To mitigate this, we augmented the generated trajectories as follows. First, we add to the domain a fluent named *dummy-predicate*, and two actions named *dummy-add-predicate-action*, and *dummy-del-predicate-action*. The "dummy" actions have no preconditions, and their effects are to add and remove the dummy predicate. Figure 6 lists the PDDL for these dummy actions.

```
(:action dummy-add-predicate-action
    :parameters (?agent - object)
    :precondition (and)
    :effect (and (dummy-predicate)
    )
)

(:action dummy-del-predicate-action
    :parameters (?agent - object)
    :precondition (and)
    :effect (and (not (dummy-predicate))
    )
)
```

Fig. 6. The new actions added to each agent's domain. The actions receive the agent as a parameter and add or delete the new predicate.

Next, we iterate over every joint action in every generated trajectory and, with probability p_{dummy} add one of the dummy actions. The added actions do not change the internal logic of the original plans, as they do not affect any of the domain's fluents and only add or delete the newly created fluent $(dummy - predicate)$. In our experiments, we set $p_{dummy} = 0.65$. As a result, at least 65% of the transitions in our trajectories, on average, include a non-trivial joint action. We chose such a high percentage of non-trivial joint actions since the focus of this research is on learning from concurrently executed actions in a multi-agent setting. Finally, we randomly selected problems and added the *dummy-predicate* to the goal state. This addition prevents a planner from solving problems unless the action model it uses includes actions that affect *dummy-predicate*, i.e., by learning from non-trivial joint actions.

For each domain, we provided the evaluated learning algorithm with at most 100 action triplets as training dataset, except in the Rovers and Overcooked domains. In these domains, all learning algorithms struggled and additional observations (action triplets) were required to learn an effective action model. To provide the additional observations for the Rovers domain, we generated single agent problems using the classical planning problem generator (Seipp et al. 2022) and solved the problems using the state-of-the-art single-agent solver Fast-Downward (FD) (Helmert 2006). The plans created by FD were then converted to multi-agent joint action plans as described previously, resulting in a training set of up to 1000 action triplets for the Rovers domain. For the Overcooked domain, we used the API by (S. A. Wu et al. 2021) to generate scenarios, resulting in 44 different problems. Similar to Rovers, we used FD to solve these problems and the training dataset included 800 triplets.

Table 1 presents characteristics of the domains used in our experiments. Columns $|A|$, $|F|$, max $arity(f)$, max $arity(a)$, "max # agents", contain the number of lifted actions, the number of lifted fluents, the maximum arity of fluents, the maximum arity of actions and the maximum number of agents acting in a problem respectively. The columns $|T|$, "%Trivial $\langle s, \hat{a}, s' \rangle$ before" and "%Trivial $\langle s, \hat{a}, s' \rangle$ after" display the number of trajectories in the domain's dataset, the percentage of the action triplets that consist only of trivial joint actions in the setting before and after adding the dummy actions, respectively. The columns avg($|\hat{a}|$) and max($|\hat{a}|$) represent the average and maximum number of actions in a joint action not including *NO-OP* or the dummy actions.

Table 1. General statistics of the domains and train trajectories in our experiments.

Domain	$\|A\|$	$\|F\|$	**max** $arity(f)$	**max** $arity(a)$	**max # agents**	$\|T\|$	**%Trivial** $\langle s, \hat{a}, s' \rangle$ **before**	**%Trivial** $\langle s, \hat{a}, s' \rangle$ **after**	avg($\|\hat{a}\|$)	max($\|\hat{a}\|$)
Blocks	4	5	2	3	4	20	90	10	1.09	2.00
Depots	5	7	3	4	12	15	92	13	1.08	2.00
Driverlog	6	6	2	4	8	20	95	10	1.04	2.00
Logistics	5	6	3	4	7	20	76	12	1.24	2.00
Rovers	9	25	2	6	10	100	88	15	1.11	2.00
Satellite	5	8	2	4	10	20	98	34	1.01	2.00
Sokoban	3	6	3	6	4	11	77	29	1.16	2.00
Overcooked	10	15	2	5	2	44	85	32	1.13	2.00

6.2 Evaluation Metrics

We evaluated our learned models using three key metrics: (1) coverage, defined as the percentage of test set problems solved using the learned action model, (2) the applicability rate actions, and (3) the precision and recall of the learned action model.

Coverage and Validation. Coverage is defined as the percentage of test set problems successfully solved using the learned action model. This metric assesses the effectiveness of the learned model in handling previously unseen scenarios. Since single-agent planners are often faster than MA-PDDL planners, we employed the state-of-the-art planner Fast-Downward (Helmert 2006), with a 15-minute time limit per problem. To validate plan correctness, we used VAL (Howey et al. 2004) with the original multi-agent domain. For verifying plans generated using MA-SAM$^+$, we applied a "reverse parsing" procedure for macro actions. This involves decomposing LMAs into their corresponding single-agent actions with appropriate bindings, and adding them sequentially to the plan. The resulting sequential plan was validated using VAL.

Applicability Rate. We also report the *applicability rate* of every action in the learned action model, denoted $App(a)$.[11] We measured this rate with respect to a set of joint action triplets $\mathcal{T}_{test}$ that are of interest. In our case, $\mathcal{T}_{test}$ was generated by solving each of the problems in our test set using M^*.

We measured $App(a)$ as follows:

$$App(a) = \frac{\sum_{\{\langle s,\hat{a},s'\rangle | a\in\hat{a}\}} app_M(a,s) \cap app_{M^*}(a,s)}{\sum_{\{\langle s,\hat{a},s'\rangle | a\in\hat{a}\}} app_{M^*}(a,s)}$$

To measure the total precision and recall for the action model, we average the values of all the actions in the domain.

Precision and Recall. Following prior work on action model learning (Aineto, Celorrio, et al. 2019; Lamanna, Saetti, et al. 2021), we measured the precision and recall of the preconditions and effects of the learned action model. This is done by measuring the rate of the redundant and missing fluents in the actions preconditions and effects in the learned action model compared to M^*. Formally, we define the precision and recall of an action's preconditions as follows:

$$P^{pre}(a) = \frac{|pre_{M^*}(a) \cap pre_M(a)|}{|pre_M(a)|}$$
$$R^{pre}(a) = \frac{|pre_{M^*}(a) \cap pre_M(\alpha)|}{|pre_{M^*}(a)|}$$

The precision and recall for the action's effects are calculated in the same way. Notice that since the learned action model is safe, then $R^{pre}(a)$ and $P^{eff}(a)$ for any action a must always be one. If an action was not learned we set its precision value to zero. Similar to the applicability metric, we average the precision and recall values over all the actions in the domain.

6.3 Results

We first present the coverage results for the evaluated algorithms. Figures 7 and 8 depicts the coverag results for all the evaluated domains. The x-axis represents the number of joint action trajectories used to learn the action model, and the y-axis represents coverage (i.e., the percent of the test set problems that were solved). As expected, MA-SAM$^+$ and MA-SAM has consistently demonstrated a significantly higher coverage than SAM across all domains compared to the other algorithms. In the Blocks, Satellite and, Sokoban domains, SAM failed to solve any of the problems in the test set, while MA-SAM$^+$ and MA-SAM were able to solve many test problems.

In all evaluated domains, MA-SAM$^+$ performs better than MA-SAM, learning an action model that can solve more problems significantly faster. This is expected, since MA-SAM$^+$ creates LMAs that enable it to perform subsets of actions MA-SAM deems unsafe. For example, in the Depots domain with 10 joint action triplets, MA-SAM achieved a coverage of 0% while MA-SAM$^+$ achieved a coverage of 73%.

Observe that in some domains, the coverage may decrease occasionally given more data. This occurs, for example, in the Blocks and Logistics domains for MA-SAM when given 50 and 55 joint action triplets respectively. In these domains, MA-SAM's coverage drops to 0%. The same decrease is also observed in the Satellite domain for MA-SAM$^+$ when given between 20 and 30 joint action triplets. In this case, we observed a drop of 45% in MA-SAM$^+$'s coverage. One reasons for this, as explained above, is that additional data may cause MA-SAM to deem some actions as not safe to apply, limiting its problem-solving capability. MA-SAM$^+$ mitigates this to some extent by creating LMAs for such cases. Another reason coverage may decline with more data is that additional actions — particularly additional LMAs— increase the complexity of planning with the learned domain. Consequently, the planner may fail to solve some test set problems within the given time constraints.

[11] Prior work (Le et al. 2024; Mordoch, Juba, et al. 2023) referred to this as the *semantic recall* of the preconditions.

Interestingly, in the Rovers domain, the coverage results plateau after 350 triplets, with both MA-SAM and MA-SAM$^+$ achieving less than 40% coverage. This is attributed to the high complexity of the domain, which contains nine lifted actions and 25 lifted fluents – the most among all domains in our dataset. Moreover, due to the high arity of the actions, each action can be bounded to a large number of possible pb-literals, resulting in overly conservative preconditions. These additional preconditions often prevent actions from being applied in states where, according to M^*, they should have been applicable. Moreover, the actions with the highest arity—communicate_image_data (arity 6), communicate_soil_data (arity 5), and communicate_rock_data (arity 5)—are almost always necessary to achieve the goal in this domain. These actions are particularly challenging for MA-SAM to learn because they have relatively few preconditions (6 out of 25 possible fluents, i.e., 50 possible literals) and only a single add-effect. A small number of true preconditions makes learning harder for MA-SAM, since the algorithm initially assumes that all literals are preconditions and depends on observations to rule out incorrect ones. In addition, the limited number of effects reduces the chances of encountering diverse states along a trajectory, which further slows the learning process.

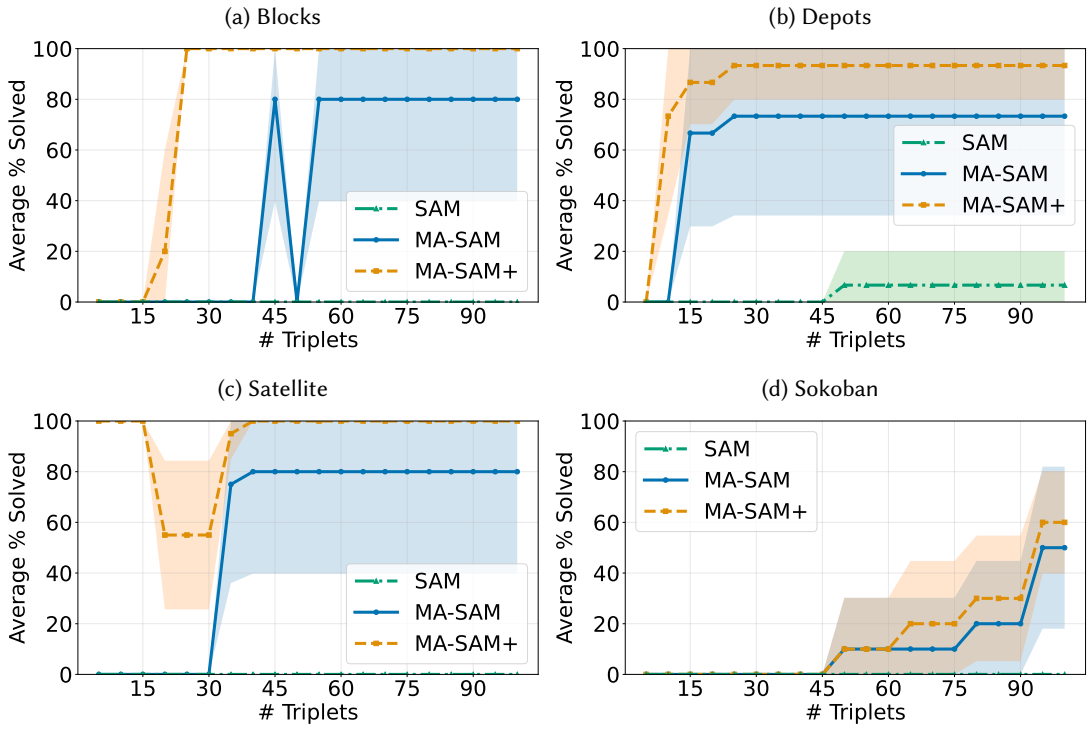

Fig. 7. Comparison of the average percentage of the problems solved as a function of the number of input joint action triplets for the domains Blocks, Depots, Satellite and Sokoban. The orange dashed line represents the MA-SAM$^+$ algorithm, the blue line corresponds to the MA-SAM algorithm, and the green line denotes the SAM algorithm.

Scalability evaluation. We measured the effect of the number of agents on the coverage obtained using the domain learned by each of the evaluated algorithms. We created new datasets for the Rovers domain, each with a different number of agents. We experimented with 2, 4, 8, and 16 agents. For this specific experiment, we did

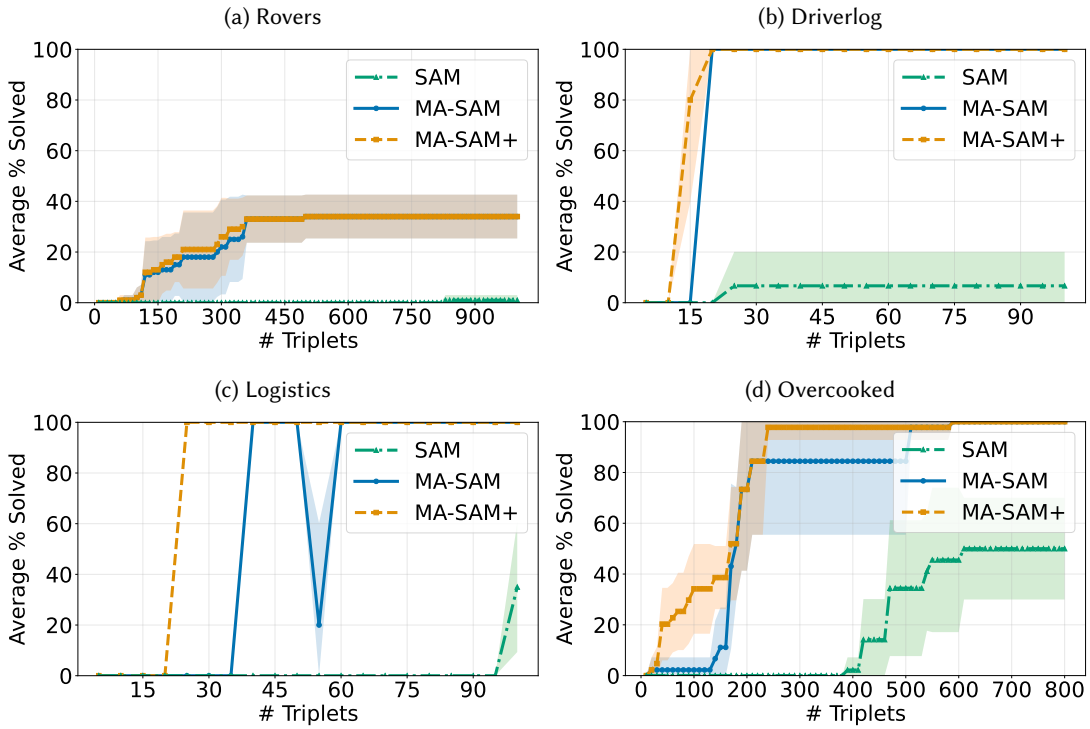

Fig. 8. Comparison of the average percentage of the problems solved as a function of the number of input joint action triplets for the domains Rovers, Driverlog, Logistics and Overcooked. The orange dashed line represents the MA-SAM$^+$ algorithm, the blue line corresponds to the MA-SAM algorithm, and the green line denotes the SAM algorithm.

not add the dummy actions, as we wanted to evaluate how only changing the number of agents in the domain affects the coverage. We used the same problem generator (Seipp et al. 2022) and generation parameters, except for varying the number of agents (rovers). Table 2 reports the number of trivial action triplets as a function of the number of agents. As expected, the number of trivial triplets decreases when more agents are present in the environment as tasks can be distributed among the agents to achieve the same goals and more actions can be performed concurrently.

Figure 9 presents the coverage results of the scalability experiments. The x-axis denotes the number of input action triplets, while the y-axis indicates the average percentage of test set problems solved and validated using the learned models. The blue, green, and orange lines correspond to the experiments with 2, 4, and 8 agents, respectively. Results for 16 agents are omitted because all algorithms could not solve any test problem. The main trend we observe is that for all algorithms in almost all cases adding more agents decrease the resulting coverage. One reason for this is that problems with more agents are harder to solve — the planner has more actions to choose from. Another reason is that the input trajectories contain joint actions with more concurrent action (as shown in Table 2), which makes the learning harder even for MA-SAM$^+$ and MA-SAM.

No notable difference was observed between MA-SAM$^+$ and MA-SAM in these results, while their advantage over SAM was observed mostly in the 4 agent experiments. The similar behavior of all algorithms in the 2 agent experiments can be explained by the high ratio of trivial joint action triplets. The similar behavior of all algorithms

in the 8 agent experiments can be explained by the increased difficulty to learn domain and solve problems when having so much more agents.

Table 2. Number of trivial actions as a function of the number of agents in the Rovers scalability experiments.

# Agents	**2**	**4**	**8**	**16**
%Trivial triplets	94	86	80	79

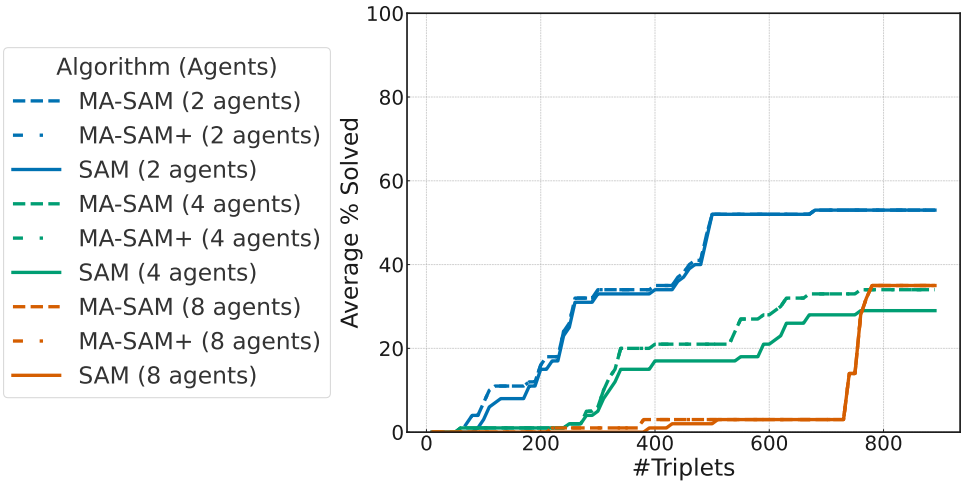

Fig. 9. Coverage results for the Rovers domain with two, four and eight agents acting in the environment. The blue, green and orange lines represent the results for the algorithms with two agents, four and eight agents respectively.

Next, we present the precision and recall results, as well as the applicability rates for the evaluated algorithms. Tables 3-10 present the precision and recall results, as well as the applicability rates for the evaluated domains. The column $|\langle s, \hat{a}, s' \rangle|$ represents the number of input joint action triplets used by the learning algorithms. The columns P^{pre}, App, and R^{eff}, display the precision of preconditions, the applicability rate of the actions in the domain, and the recall of effects respectively. The best-performing algorithm for each evaluated metric, given the number of input joint action triplets, is highlighted in bold. The results are provided until convergence, i.e., until the point they no longer change before reaching the maximum number of triplets we provided for the domain.

In all our experimented domains, we observed that MA-SAM significantly outperformed SAM in all evaluated metrics. For example, in the Blocks domain (Table 3), with 70 joint action triplets, MA-SAM achieved P^{pre} of 0.52 and App of 0.86 while SAM only achieved 0.21 and 0.38 respectively. This is expected, as SAM only learns from trivial joint action triplets. Thus, it utilizes significantly fewer joint action triplets compared to MA-SAM, as can be seen in the column "%Trivial triplets after" in Table 1.

One may be concerned by the low precision values of the preconditions (P^{pre}). This indicates that the learned action model has much more preconditions than in M^*. However, as shown by the significantly higher values of App, the additional preconditions do not severely affect the actions' applicability according to the learned action model. For example, in the Logistics domain (Table 4), with 100 joint action triplets, P^{pre} is 0.66, but App is 0.99 indicating that while the actions' preconditions precision is relatively low, the actions were applicable in

nearly all of the tested triplets. In most domains, this gap occurs because many of the redundant preconditions learned by MA-SAM are, in fact, implicitly correct in the domain, e.g., negative preconditions that are due to mutex relations. In the Blocks domain, for example, the *pick-up* action implicitly requires that the agent is not holding a block prior to the action's execution, i.e., the fluent $(\neg(holding\ ?a\ ?x)$ is implicitly true whenever the action is executed.

We also observed that in some domains, the performance of MA-SAM does not improve monotonically with the number of given triplets. In the Blocks domain (Table 3), when moving from 45 to 50 joint action triplet, the P^{pre} and App of MA-SAM drop from 0.35 to 0.32 and from 0.68 to 0.65, respectively. This occurs because the set of actions that can be safely applied (Def. 4.1) may be reduced as more observations are given. Specifically, we observed that *stack* was considered safe to apply when MA-SAM was given 45 joint action triplets, having *dummy-predicate* as a precondition. However, in the next 5 triplets MA-SAM observed a joint action triplet $\langle s, \hat{a}, s' \rangle$ where $\hat{a}$ includes *stack* and *dummy-add-predicate-action*, indicating that they were performed concurrently. The fluent *dummy-predicate* was not in s but was in s'. Therefore, MA-SAM removed it from the preconditions of *stack* and added the non-unit clause IsEff((*dummy-predicate*), *stack*) ∨ IsEff((*dummy-predicate*),*dummy-add-predicate-action*). As a result, MA-SAM removed *stack* from the action model it returned, resulting in lower P^{pre} and App. In contrast, because SAM does not observe joint actions, its performance increases monotonically. Similar trends were observed in the Logistics and Satellite domains (Table 4 and Table 8).

In the Rovers domain (Table 7), we required significantly more joint action triplets to achieve performance comparable to other domains. Specifically, with 1000 joint action triplets we observed similar performance to that observed in the other domains after less than 100 joint action triplets. This outcome is attributed to the inherent complexity of the Rovers domain, which contains a significantly larger number of actions and fluents compared to the other domains (see Table 1). We also observed similar trends in the Overcooked domain (Table 10), as the domain includes the highest number of actions.

Finally, notice that in the Blocks, Depots, Driverlog, Logistics, and Rovers domains the effects recall values were constantly one for both MA-SAM and SAM. This indicates that, in these domains, if an action was observed, its effects are learned perfectly after a single observation. We observed that in these domains the add effects are implicitly required to be false prior to the actions execution. In addition, in these domains, delete effects are required as preconditions, resulting in them being observed every time the action is executed. In contrast, in the Satellite, Sokoban and Overcooked domains (Tables 8, 9 and 10), there are actions where the delete effects are not required as preconditions. This affected the R^{eff} values. For example, the actions *switch_on*, *push-to-nongoal*, and *move* in the Satellite, Sokoban and Overcooked domains respectively have a delete effects $(not\ (calibrated\ ?i))$, $(not\ (at-goal\ ?s))$ and $(not\ (occupied\ ?start))$ respectively, which are not required as preconditions.

Summary. When evaluating the effectiveness of the domain in terms of coverage, we observed that MA-SAM$^+$ and MA-SAM are most effective in scenarios with high concurrency, with MA-SAM$^+$ demonstrating superior performance, particularly when ambiguities remain unresolved. Moreover, across all domains, MA-SAM significantly outperformed SAM in terms of precision, recall and applicability.

7 Discussion: Beyond Independent Actions

MA-SAM and MA-SAM$^+$ are designed for cases where single-agent actions are well-defined. In some domains, however, actions may not be well-defined and behave differently when executed concurrently (Boutilier and R. I. Brafman 2001). This section discusses how MA-SAM and MA-SAM$^+$ can be extended to support such domains. Such domains may include *conflicting actions*, i.e., where the preconditions of performing single-agent actions include constraints over which other single-agent actions are performed concurrently, as well as *collaborative actions*. Recall, collaborative actions are sets of actions whose effects differ from the union of their constituent single-agent actions. The SAMoJA algorithm, defined in Section 5, can be directly applied for such domains.

Table 3. Comparison of the precision and recall and applicability rates for MA-SAM and SAM for the Blocks domain.

	P^{pre}		App		R^{eff}	
$\lvert \langle s, \hat{a}, s' \rangle \rvert$	**MA-SAM**	**SAM**	**MA-SAM**	**SAM**	**MA-SAM**	**SAM**
5	**0.11**	0.00	**0.25**	0.00	1.00	1.00
10	**0.11**	0.00	**0.25**	0.00	1.00	1.00
15	**0.14**	0.00	**0.28**	0.00	1.00	1.00
20	**0.17**	0.00	**0.33**	0.00	1.00	1.00
25	**0.17**	0.00	**0.41**	0.00	1.00	1.00
30	**0.17**	0.00	**0.41**	0.00	1.00	1.00
35	**0.17**	0.00	**0.41**	0.00	1.00	1.00
40	**0.17**	0.00	**0.41**	0.00	1.00	1.00
45	**0.35**	0.03	**0.68**	0.09	1.00	1.00
50	**0.32**	0.03	**0.65**	0.09	1.00	1.00
55	**0.50**	0.09	**0.84**	0.11	1.00	1.00
60	**0.52**	0.18	**0.86**	0.15	1.00	1.00
65	**0.52**	0.20	**0.86**	0.26	1.00	1.00
70	**0.52**	0.21	**0.86**	0.38	1.00	1.00

Table 4. Comparison of the precision and recall and applicability rates for MA-SAM and SAM for the Logistics domain.

	P^{pre}		App		R^{eff}	
$\lvert \langle s, \hat{a}, s' \rangle \rvert$	**MA-SAM**	**SAM**	**MA-SAM**	**SAM**	**MA-SAM**	**SAM**
5	**0.10**	0.00	**0.19**	0.00	1.00	1.00
10	**0.10**	0.00	**0.19**	0.00	1.00	1.00
15	**0.14**	0.00	**0.29**	0.00	1.00	1.00
20	**0.19**	0.00	**0.39**	0.00	1.00	1.00
25	**0.25**	0.00	**0.49**	0.00	1.00	1.00
30	**0.25**	0.00	**0.49**	0.00	1.00	1.00
35	**0.25**	0.00	**0.49**	0.00	1.00	1.00
40	**0.57**	0.01	**0.88**	0.02	1.00	1.00
45	**0.60**	0.02	**0.90**	0.03	1.00	1.00
50	**0.61**	0.07	**0.93**	0.05	1.00	1.00
55	**0.60**	0.13	**0.86**	0.15	1.00	1.00
60	**0.62**	0.16	**0.96**	0.20	1.00	1.00
65	**0.66**	0.21	**0.99**	0.22	1.00	1.00
70	**0.66**	0.22	**0.99**	0.24	1.00	1.00
75	**0.66**	0.27	**0.99**	0.26	1.00	1.00
80	**0.66**	0.32	**0.99**	0.42	1.00	1.00
85	**0.66**	0.33	**0.99**	0.44	1.00	1.00
90	**0.66**	0.33	**0.99**	0.44	1.00	1.00
95	**0.66**	0.34	**0.99**	0.52	1.00	1.00
100	**0.66**	0.37	**0.99**	0.55	1.00	1.00

SAMoJA cannot scale beyond a small number of agents since the number of joint actions grows exponentially with the number of agents. However, several natural restrictions over the relation between joint actions and the corresponding single-agent actions can allow more efficient learning.

Table 5. Comparison of the precision and recall and applicability rates for MA-SAM and SAM for the Depots domain.

	P^{pre}		App		R^{eff}	
$\lvert \langle s, \hat{a}, s' \rangle \rvert$	**MA-SAM**	**SAM**	**MA-SAM**	**SAM**	**MA-SAM**	**SAM**
5	**0.16**	0.00	**0.25**	0.00	1.00	1.00
10	**0.33**	0.04	**0.53**	0.04	1.00	1.00
15	**0.56**	0.06	**0.81**	0.07	1.00	1.00
20	**0.57**	0.21	**0.84**	0.30	1.00	1.00
25	**0.59**	0.25	**0.87**	0.33	1.00	1.00
30	**0.59**	0.30	**0.87**	0.37	1.00	1.00

Table 6. Comparison of the precision and recall and applicability rates for MA-SAM and SAM for the Driverlog domain.

	P^{pre}		App		R^{eff}	
$\lvert \langle s, \hat{a}, s' \rangle \rvert$	**MA-SAM**	**SAM**	**MA-SAM**	**SAM**	**MA-SAM**	**SAM**
5	**0.25**	0.10	**0.33**	0.09	1.00	1.00
10	**0.33**	0.12	**0.54**	0.16	1.00	1.00
15	**0.39**	0.17	**0.57**	0.25	1.00	1.00
20	**0.54**	0.22	**0.77**	0.33	1.00	1.00
25	**0.54**	0.23	**0.77**	0.33	1.00	1.00
30	**0.55**	0.25	**0.79**	0.35	1.00	1.00
35	**0.56**	0.25	**0.81**	0.35	1.00	1.00
40	**0.57**	0.25	**0.85**	0.35	1.00	1.00
45	**0.58**	0.25	**0.88**	0.35	1.00	1.00
50	**0.59**	0.25	**0.90**	0.35	1.00	1.00
55	**0.59**	0.26	**0.90**	0.36	1.00	1.00
60	**0.59**	0.27	**0.90**	0.37	1.00	1.00
65	**0.60**	0.27	**0.93**	0.40	1.00	1.00
70	**0.60**	0.27	**0.93**	0.40	1.00	1.00
75	**0.60**	0.27	**0.93**	0.42	1.00	1.00
80	**0.60**	0.27	**0.93**	0.45	1.00	1.00

Conflicting Actions. Consider domains with conflicting but not collaborative actions. In such domains, MA-SAM and MA-SAM$^+$ can be used to learn actions' effects, but learning preconditions is challenging. Without any restriction on the types of conflicts that may occur, we cannot make assumptions about the safety of joint actions that were not observed in the given trajectories. Thus, memorizing the previously observed actions is the only way to learn a safe action model.

More efficient learning can be achieved by restricting the types of possible conflicts. For example, if conflicts are defined for at most n concurrently executed actions, then we only need to memorize joint actions of up to size n, and can then generalize to joint actions containing more single-agent actions. For example, given three actions a_1, a_2, a_3 and $n = 2$ if we observe the joint actions: $[a_1, a_2]$, $[a_1, a_3]$, $[a_2, a_3]$, we can infer that the joint execution of $[a_1, a_2, a_3]$ is safe. Assuming $k \gg n$ agents, this approach significantly reduces learning complexity while maintaining the safety property.

Another type of conflict that may arise is caused by violating the concurrency constraint (Boutilier and R. I. Brafman 2001). A concurrency constraint prohibits some joint actions, either because two or more actions cannot be performed in parallel or, on the contrary, because some actions must be executed in parallel (Boutilier and R. I. Brafman 2001; Crosby, Jonsson, et al. 2014). Under such a setting, we can learn an approximation to the

Table 7. Comparison of the precision and recall and applicability rates for MA-SAM and SAM for the Rovers domain.

	P^{pre}		App		R^{eff}	
$\lvert \langle s, \hat{a}, s' \rangle \rvert$	**MA-SAM**	**SAM**	**MA-SAM**	**SAM**	**MA-SAM**	**SAM**
10	**0.14**	0.06	**0.05**	0.01	1.00	1.00
50	**0.39**	0.09	**0.53**	0.01	1.00	1.00
100	**0.51**	0.17	**0.69**	0.09	1.00	1.00
150	**0.53**	0.20	**0.74**	0.17	1.00	1.00
200	**0.56**	0.28	**0.80**	0.37	1.00	1.00
300	**0.56**	0.31	**0.80**	0.46	1.00	1.00
400	**0.60**	0.33	**0.85**	0.50	1.00	1.00
500	**0.61**	0.35	**0.85**	0.52	1.00	1.00
600	**0.61**	0.36	**0.85**	0.57	1.00	1.00
700	**0.61**	0.36	**0.85**	0.58	1.00	1.00
800	**0.61**	0.36	**0.85**	0.58	1.00	1.00
900	**0.61**	0.37	**0.85**	0.58	1.00	1.00
1000	**0.61**	0.38	**0.86**	0.59	1.00	1.00

Table 8. Comparison of the precision and recall and applicability rates for MA-SAM and SAM for the Satellite domain.

	P^{pre}		App		R^{eff}	
$\lvert \langle s, \hat{a}, s' \rangle \rvert$	**MA-SAM**	**SAM**	**MA-SAM**	**SAM**	**MA-SAM**	**SAM**
5	**0.22**	0.00	**0.28**	0.00	0.95	**1.00**
10	**0.22**	0.00	**0.28**	0.00	0.95	**1.00**
15	**0.22**	0.00	**0.28**	0.00	0.95	**1.00**
20	**0.18**	0.00	**0.25**	0.00	0.99	**1.00**
25	**0.18**	0.00	**0.25**	0.00	0.99	**1.00**
30	**0.18**	0.00	**0.25**	0.00	0.99	**1.00**
35	**0.43**	0.14	**0.67**	0.04	0.95	**0.96**
40	**0.59**	0.20	**0.83**	0.12	0.96	0.96
45	**0.59**	0.23	**0.83**	0.23	0.96	0.96
50	**0.61**	0.23	**0.87**	0.23	**1.00**	0.96
55	**0.61**	0.24	**0.87**	0.32	**1.00**	0.96
60	**0.61**	0.24	**0.87**	0.32	**1.00**	0.96
65	**0.61**	0.24	**0.87**	0.32	**1.00**	0.96
70	**0.60**	0.24	**0.85**	0.32	**1.00**	0.96

lower and upper bounds for the concurrency constraint of the actions' objects. This can be achieved by assigning a lower and upper bound on the actions' parameters based on the observations in which the actions are being applied. I.e., for each joint action, we can calculate the number of actions that interact with the same objects and calculate a legal execution interval. That way, actions will only be applicable if they conform with their concurrency constraint. The sample complexity presented in Theorem 3 also addresses actions with concurrency constraints.

The last type of conflict we discuss is based on ill-defined joint actions (Boutilier and R. I. Brafman 2001). These are joint actions in which some of the actions' preconditions or effects may conflict. We propose a conflict inference stage for MA-SAM to address this setting. This conflict inference is conducted by grounding the learned actions during the learning stage, using the observation objects, and inferring whether two grounded actions have contradicting preconditions or effects. If so, we can deduce that these two actions cannot be executed in

Table 9. Comparison of the precision and recall and applicability rates for MA-SAM and SAM for the Sokoban domain.

	P^{pre}		App		R^{eff}	
$\lvert \langle s, \hat{a}, s' \rangle \rvert$	**MA-SAM**	**SAM**	**MA-SAM**	**SAM**	**MA-SAM**	**SAM**
5	**0.06**	0.02	**0.05**	0.05	0.97	**0.99**
10	**0.15**	0.02	**0.23**	0.05	0.97	**0.99**
15	**0.19**	0.02	**0.27**	0.05	0.97	**0.99**
20	**0.37**	0.09	**0.49**	0.24	0.97	**0.98**
25	**0.37**	0.09	**0.50**	0.24	0.98	0.98
30	**0.50**	0.09	**0.56**	0.24	0.98	0.98
35	**0.50**	0.09	**0.56**	0.25	0.98	0.98
40	**0.50**	0.09	**0.57**	0.25	**0.99**	0.98
45	**0.51**	0.09	**0.65**	0.25	**0.99**	0.98
50	**0.52**	0.09	**0.66**	0.25	**0.99**	0.98
55	**0.52**	0.09	**0.66**	0.25	**0.99**	0.98
60	**0.52**	0.09	**0.70**	0.25	**0.99**	0.98
65	**0.52**	0.09	**0.70**	0.25	**0.99**	0.98
70	**0.53**	0.09	**0.71**	0.25	**0.99**	0.98
75	**0.53**	0.09	**0.71**	0.25	**0.99**	0.98
80	**0.53**	0.09	**0.72**	0.25	**0.99**	0.98
85	**0.53**	0.09	**0.72**	0.25	**0.99**	0.98
90	**0.53**	0.09	**0.72**	0.25	**0.99**	0.98
95	**0.54**	0.09	**0.75**	0.25	**0.99**	0.98

Table 10. Comparison of the precision and recall and applicability rates for MA-SAM and SAM for the Overcooked domain.

	P^{pre}		App		R^{eff}	
$\lvert \langle s, \hat{a}, s' \rangle \rvert$	**MA-SAM**	**SAM**	**MA-SAM**	**SAM**	**MA-SAM**	**SAM**
10	**0.15**	0.14	**0.22**	0.05	**0.35**	0.15
100	**0.38**	0.21	**0.63**	0.28	**0.72**	0.50
200	**0.42**	0.24	**0.86**	0.40	**0.87**	0.65
300	**0.42**	0.24	**0.87**	0.49	**0.87**	0.71
400	**0.42**	0.25	**0.89**	0.53	**0.88**	0.73
500	**0.42**	0.25	**0.89**	0.60	**0.88**	0.78
600	**0.43**	0.25	**0.91**	0.61	**0.90**	0.78
700	**0.43**	0.25	**0.91**	0.63	**0.90**	0.78
800	**0.43**	0.26	**0.91**	0.65	**0.90**	0.78

parallel while interacting with the same objects and should always be independent. We can use this insight to guide solvers by pruning invalid concurrent executions and thus reducing the branching factor.

Collaborative Actions. Since unobserved collaborative effects cannot be inferred, to safely learn collaborative actions, we must observe all possible joint executions of actions. However, learning labeled collaborative actions such as those presented by (Shekhar and R. I. Brafman 2020) is already supported in MA-SAM$^+$: such actions are additional macro actions with the agents as part of their parameters. For unlabeled collaborative actions, we observe that collaborative effects are, in fact, conditional effects of the joint actions. When the additional "condition" stating the single-agent actions composing the joint action interact with the same object holds, then the resulting collaborative effect will manifest. To learn such collaborative conditional effects, we require a bound

n on the number of actions in a joint action that can create collaborative effects. Then, we use MA-SAM$^+$ to learn all joint actions up to size n as presented by (Mordoch, Scala, et al. 2024). Applying MA-SAM$^+$ while learning the conditional effects will output new macro actions with conditional effects similar to those presented by (Hofmann et al. 2020).

8 Conclusion and Future Work

This work explored the problem of learning action models in a multi-agent setting, namely in environments that can be represented in MA-PDDL (Kovacs 2012). Unlike previous work (Zhuo, Muñoz-Avila, et al. 2011), we learned the single agents' action models from trajectories including joint action triplets. We presented MA-SAM, an algorithm capable of learning from such trajectories. This algorithm has a runtime that is linear in the agents, lifted actions, and action triplets, and quadratic in the number of pb-literals. MA-SAM is guaranteed to return a safe action model, i.e., one that guarantees all actions can be safely applied according to the real, unknown action model. We also analyzed and provided a lower bound on MA-SAM's sample complexity. We observed that in some cases, MA-SAM encounters ambiguities it cannot resolve. To resolve these ambiguities, we presented SAMoJA, which cannot scale but has a bounded sample complexity. Then, we presented MA-SAM$^+$, which extends MA-SAM by learning Lifted Macro Actions (LMAs) in cases where it cannot safely identify single-agent actions' effects. MA-SAM$^+$ is also guaranteed to return a safe action model, but its worst case runtime is exponential in the number of agents.

To evaluate MA-SAM and MA-SAM$^+$, we experimented with seven classical multi-agent planning benchmarks, as well as the novel benchmark domain, *Overcooked*. We showed that when most actions are observed to be executed concurrently, MA-SAM outperforms SAM in terms of model precision and recall, as well as its ability to solve problems. We also showed that in cases where MA-SAM could not disambiguate on the actions' effects, MA-SAM$^+$ could learn LMAs and thus solve more of the test set problems. Finally, we discussed different approaches to learning safe action models when actions may interact. Future work will integrate MA-SAM in more complex environments with agents possibly having collaborative or conflicting actions. In addition, we aim to integrate MA-SAM with multi-agent reinforcement learning techniques, and run it in an online manner.

Yet another direction for future work is to generalize to other MAP formalisms beyond MA-PDDL. Two such formalisms are *Multi-Agent Hierarchical Task Network (MA-HTN)* (Clement and Durfee 1999) and *Decentralized Partially Observable Markov Decision Processes (Dec-POMDPs)* (Bernstein et al. 2002). MA-HTN is a multi-agent extension of the single-agent HTN formalism. It includes information about how abstract plans can be refined to identify and avoid potential conflicts in the multi-agent setting. Dec-POMDP is a multi-agent extension of the classical POMDP formalism where multiple agents act in the same environment. A key difference between POMDP and Dec-POMDP is that different agents may receive different observations during execution. Learning action models for such more complex formalism is worthwhile as they are more expressive than MA-PDDL.

Acknowledgments

This research was partly supported by the Israel Science Foundation (ISF) grant #1238/23 awarded to Roni Stern, and by the Magneton Program of the Israel Innovation Authority (Grant No. 90011). Brendan Juba was supported by NSF award IIS-1942336.

References

D. Aineto, S. J. Celorrio, and E. Onaindia. 2019. "Learning action models with minimal observability." *Artificial Intelligence*, 275, 104–137.

D. Aineto, S. Jiménez, and E. Onaindia. 2022. "A comprehensive framework for learning declarative action models." *Journal of Artificial Intelligence Research*, 74, 1091–1123.

E. Amir and A. Chang. 2008. "Learning partially observable deterministic action models." *Journal of Artificial Intelligence Research*, 33, 349–402.

M. Asai and A. Fukunaga. 2018. "Classical planning in deep latent space: Bridging the subsymbolic-symbolic boundary." In: *Proceedings of the aaai conference on artificial intelligence* 1. Vol. 32, 6094–6101.

K. Azadeh, R. De Koster, and D. Roy. 2019. "Robotized and automated warehouse systems: Review and recent developments." *Transportation Science*, 53, 4, 917–945.

S. Barrett and P. Stone. 2012. "An analysis framework for ad hoc teamwork tasks." In: *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 357–364.

D. S. Bernstein, R. Givan, N. Immerman, and S. Zilberstein. 2002. "The complexity of decentralized control of Markov decision processes." *Mathematics of operations research*, 27, 4, 819–840.

D. Borrajo. 2013. "Multi-agent planning by plan reuse." In: *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*, 1141–1142.

A. Botea, M. Enzenberger, M. Müller, and J. Schaeffer. 2005. "Macro-FF: Improving AI planning with automatically learned macro-operators." *Journal of Artificial Intelligence Research*, 24, 581–621.

C. Boutilier and R. I. Brafman. 2001. "Partial-order planning with concurrent interacting actions." *Journal of Artificial Intelligence Research*, 14, 105–136.

R. I. Brafman. 2015. "A privacy preserving algorithm for multi-agent planning and search." *Distributed and Multi-Agent Planning (DMAP-15)*, 1.

R. I. Brafman and C. Domshlak. 2013. "On the complexity of planning for agent teams and its implications for single agent planning." *Artificial Intelligence*, 198, 52–71.

B. J. Clement and E. H. Durfee. 1999. "Top-down search for coordinating the hierarchical plans of multiple agents." In: *Proceedings of the third annual conference on Autonomous Agents*, 252–259.

S. Cresswell and P. Gregory. 2011. "Generalised domain model acquisition from action traces." In: *International Conference on Automated Planning and Scheduling (ICAPS)*, 42–49.

S. N. Cresswell, T. L. McCluskey, and M. M. West. 2013. "Acquiring planning domain models using LOCM." *The Knowledge Engineering Review*, 28, 2, 195–213.

M. Crosby, A. Jonsson, and M. Rovatsos. 2014. "A Single-Agent Approach to Multiagent Planning." In: *ECAI*, 237–242.

M. Crosby, M. Rovatsos, and R. Petrick. 2013. "Automated agent decomposition for classical planning." In: *Proceedings of the International Conference on Automated Planning and Scheduling*. Vol. 23, 46–54.

R. E. Fikes and N. J. Nilsson. 1971. "STRIPS: A new approach to the application of theorem proving to problem solving." *Artificial intelligence*, 2, 3-4, 189–208.

M. Helmert. 2006. "The fast downward planning system." *Journal of Artificial Intelligence Research*, 26, 191–246.

T. Hofmann, T. Niemueller, and G. Lakemeyer. 2020. "Macro operator synthesis for adl domains." In: *ECAI 2020*. IOS Press, 761–768.

R. Howey, D. Long, and M. Fox. 2004. "VAL: Automatic plan validation, continuous effects and mixed initiative planning using PDDL." In: *16th IEEE International Conference on Tools with Artificial Intelligence*. IEEE, 294–301.

B. Juba, H. S. Le, and R. Stern. 2021. "Safe Learning of Lifted Action Models." In: *International Conference on Principles of Knowledge Representation and Reasoning (KR)*, 379–389.

B. Juba and R. Stern. 2022. "Learning Probably Approximately Complete and Safe Action Models for Stochastic Worlds." In: *AAAI*, 9795–9804.

P. C. Kabongo, T. M. F. Ramos, A. F. Leite, C. G. Ralha, and L. Weigang. 2016. "A multi-agent planning model for airport ground handling management." In: *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2354–2359.

R. A. Knepper, T. Layton, J. Romanishin, and D. Rus. 2013. "Ikeabot: An autonomous multi-robot coordinated furniture assembly system." In: *2013 IEEE International conference on robotics and automation*. IEEE, 855–862.

A. Komenda, M. Stolba, and D. L. Kovacs. 2016. "The international competition of distributed and multiagent planners (CoDMAP)." *AI Magazine*, 37, 3, 109–115.

R. E. Korf. 1985. "Macro-operators: A weak method for learning." *Artificial intelligence*, 26, 1, 35–77.

D. L. Kovacs. 2012. "A Multi-Agent Extension of PDDL3.1." In: *Workshop on the International Planning Competition (IPC) in the International Conference on Automated Planning and Scheduling (ICAPS)*, 19–27.

L. Lamanna, A. Saetti, L. Serafini, A. Gerevini, and P. Traverso. 2021. "Online Learning of Action Models for PDDL Planning." In: *IJCAI*, 4112–4118.

L. Lamanna and L. Serafini. 2024. "Action Model Learning from Noisy Traces: a Probabilistic Approach." In: *ICAPS*. AAAI Press, 342–350.

H. S. Le, B. Juba, and R. Stern. 2024. "Learning Safe Action Models with Partial Observability." In: *AAAI Conference on Artificial Intelligence* 18. Vol. 38, 20159–20167.

S. Maliah, G. Shani, and R. Stern. 2016a. "Collaborative privacy preserving multi-agent planning." *Autonomous Agents and Multi-Agent Systems*, 1–38.

S. Maliah, G. Shani, and R. Stern. 2017. "Collaborative privacy preserving multi-agent planning." *Autonomous Agents and Multi-Agent Systems*, 31, 3, 493–530.

S. Maliah, G. Shani, and R. Stern. 2014. "Privacy preserving landmark detection." In: *ECAI 2014*. IOS Press, 597–602.

S. Maliah, G. Shani, and R. Stern. 2016b. "Stronger privacy preserving projections for multi-agent planning." In: *Proceedings of the International Conference on Automated Planning and Scheduling*. Vol. 26, 221–229.
D. McDermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, and D. Wilkins. 1998. *PDDL-the planning domain definition language*. (1998).
A. Mordoch, B. Juba, and R. Stern. 2023. "Learning Safe Numeric Action Models." In: *AAAI Conference on Artificial Intelligence*, 12079–12086.
A. Mordoch, E. Scala, R. Stern, and B. Juba. 2024. "Safe learning of pddl domains with conditional effects." In: *Proceedings of the International Conference on Automated Planning and Scheduling*. Vol. 34, 387–395.
R. Nissim and R. Brafman. 2014. "Distributed heuristic forward search for multi-agent planning." *Journal of Artificial Intelligence Research*, 51, 293–332.
S. D. Ramchurn, F. Wu, W. Jiang, J. E. Fischer, S. Reece, S. Roberts, T. Rodden, C. Greenhalgh, and N. R. Jennings. 2016. "Human–agent collaboration for disaster response." *Autonomous Agents and Multi-Agent Systems*, 30, 1, 82–111.
J. Seipp, Á. Torralba, and J. Hoffmann. 2022. *PDDL Generators*. https://doi.org/10.5281/zenodo.6382173. (2022).
S. Shekhar and R. I. Brafman. 2020. "Representing and planning with interacting actions and privacy." *Artificial Intelligence*, 278, 103200.
R. Stern and B. Juba. 2017. "Efficient, Safe, and Probably Approximately Complete Learning of Action Models." In: *the International Joint Conference on Artificial Intelligence (IJCAI)*, 4405–4411.
M. Štolba, A. Komenda, and D. Kovacs. 2016. "Competition of distributed and multiagent planners (codmap)." In: *AAAI Conference on Artificial Intelligence*, 4343–4345.
M. Tan. 1993. "Multi-agent reinforcement learning: Independent vs. cooperative agents." In: *Proceedings of the tenth international conference on machine learning*, 330–337.
A. Torreno, E. Onaindia, A. Komenda, and M. Štolba. 2017. "Cooperative multi-agent planning: A survey." *ACM Computing Surveys (CSUR)*, 50, 6, 1–32.
J. Tožička, J. Jakubův, A. Komenda, and M. Pěchouček. 2016. "Privacy-concerned multiagent planning." *Knowledge and Information Systems*, 48, 581–618.
J. Tožička, M. Štolba, and A. Komenda. 2017. "The limits of strong privacy preserving multi-agent planning." In: *Proceedings of the International Conference on Automated Planning and Scheduling*. Vol. 27, 297–305.
P. Verma, S. R. Marpally, and S. Srivastava. 2021. "Asking the right questions: Learning interpretable action models through query answering." In: *Proceedings of the AAAI Conference on Artificial Intelligence* 13. Vol. 35, 12024–12033.
A. Wachi, Y. Sui, Y. Yue, and M. Ono. 2018. "Safe exploration and optimization of constrained MDPs using Gaussian processes." In: *AAAI Conference on Artificial Intelligence (AAAI)*, 6548–6555.
K. Wu, Q. Yang, and Y. Jiang. 2007. "ARMS: An automatic knowledge engineering tool for learning action models for AI planning." *The Knowledge Engineering Review*, 22, 2, 135–152.
S. A. Wu, R. E. Wang, J. A. Evans, J. B. Tenenbaum, D. C. Parkes, and M. Kleiman-Weiner. 2021. "Too many cooks: Coordinating multi-agent collaboration through inverse planning." *Topics in Cognitive Science*, n/a, n/a. doi:https://doi.org/10.1111/tops.12525.
K. Xi, S. Gould, and S. Thiébaux. 2024. "Neuro-Symbolic Learning of Lifted Action Models from Visual Traces." In: *Proceedings of the International Conference on Automated Planning and Scheduling*. Vol. 34, 653–662.
Q. Yang, K. Wu, and Y. Jiang. 2007. "Learning action models from plan examples using weighted MAX-SAT." *Artificial Intelligence*, 171, 2-3, 107–143.
H. H. Zhuo and S. Kambhampati. 2013. "Action-model acquisition from noisy plan traces." In: *International Joint Conference on Artificial Intelligence (IJCAI)*, 2444–2450.
H. H. Zhuo, H. Muñoz-Avila, and Q. Yang. 2011. "Learning action models for multi-agent planning." In: *International Conference on Autonomous Agents and Multiagent Systems*, 217–224.

Received 21 September 2025; accepted 28 March 2026