Knowl Inf Syst (2016) 48:581–618
DOI 10.1007/s10115-015-0887-7

REGULAR PAPER

# Privacy-concerned multiagent planning

**Jan Tožička**[1] · **Jan Jakubův**[1] · **Antonín Komenda**[1] · **Michal Pěchouček**[1]

Received: 14 January 2015 / Revised: 9 June 2015 / Accepted: 5 October 2015 /
Published online: 14 October 2015

**Abstract** Coordinated sequential decision making of a team of cooperative agents can be described by principles of multiagent planning. Provided that the mechanics of the environment the agents act in is described as a deterministic transitions system, an appropriate planning model is MA-STRIPS. Multiagent planning modeled as MA-STRIPS prescribes exactly what information has to be kept private and which information can be communicated in order to coordinate toward shared or individual goals. We propose a multiagent planning approach which combines compilation for a classical state-of-the-art planner together with a compact representation of local plans in the form of finite-state machines. Proving soundness and completeness of the approach, the planner efficiency is further boosted up using distributed delete-relaxation heuristics and using an approximative local plan analysis. We experimentally evaluate applicability of our approach in full privacy setting where only public information can be communicated. We analyze properties of standard multiagent benchmarks from the perspective of classification of private and public information. We show that our approach can be used with different privacy settings and that it outperforms state-of-the-art planners designed directly for particular privacy classification.

**Keywords** Multiagent planning · Finite-state machines · Delete relaxation · Action landmarks

✉ Jan Tožička
jan.tozicka@agents.fel.cvut.cz; tozicka@agents.fel.cvut.cz

Jan Jakubův
jakubuv@agents.fel.cvut.cz

Antonín Komenda
komenda@agents.fel.cvut.cz

Michal Pěchouček
pechoucek@agents.fel.cvut.cz

[1] Faculty of Electrical Engineering, Czech Technical University, Technická 2, 166 27 Prague, Czech Republic

## 1 Introduction

A team of intelligent agents acting in a shared environment has to coordinate its steps in order to achieve common goals. Multiagent planning with a deterministic model describes such problems and proposes techniques to solve them. Provided that the agents are obliged to keep information about their individual abilities private, they are not allowed to communicate it with other agents. Consider an application domain in which several logistic companies have to cooperate to fulfill complex transportation tasks, which cannot be managed by any of the companies on its own. Although the companies have to cooperate, still they need to keep their know-how secret, being it their modes of transportation or local routes. In such a situation, the companies represented by planning agents would not benefit from any competitive behavior as the objective is common for all the companies, but they still have to keep parts of their knowledge private because of their local competition.

The concept of private knowledge is not new in multiagent planning because it can factor a planning problem and thus positively affect the complexity of the planning process [6]. Nevertheless, multiagent planners usually do not target this particular facet of the problem. Some multiagent state-space search algorithms avoid this problem by obfuscating or aggregating private information on interconnecting states. In contrast, our coordination-centric approach completely preserves private knowledge as only public projections of agent plans are exchanged. Moreover, our algorithm benefits from decrease of required communication among the agents when compared to state-space search algorithms.

The most used multiagent planning model MA-Strips [6] prescribes a scheme which information has to stay public. Although the original motivation was not to jeopardize completeness of the planning process required for complexity assurances, most of the planners in the literature stick to this particular definition. The most notable exception is the FMAP planner [33] which allows to mark public information in the planning problem description. Besides representation of local plans as totally or partially ordered sequences of actions, a compact representation of set of local plans utilizing various types of finite-state machines was proposed in [12] and our recent work [36]. In [35], we have proposed notions of *external actions* and public plan *extensibility*. When planning with external actions, agents are informed about public actions of other agents. Hence, they are able to plan actions *for* other agents. However, external actions are striped of private information, and thus, it can happen that an agent plans an external action inappropriately. The notion of extensibility allows to recognize plans where external actions are used correctly. In [36], we have used extensibility with finite-state machines to outline a generic scheme of multiagent planners further elaborated in this work. In [17], we have improved an extensibility-based planner with a type system checker for process calculi utilized to efficiently approximate plan extensibility.

In this article, we present an extensibility-based multiagent planning algorithm which utilizes finite-state machines to compactly represent sets of plans. We call this representation *planning state machines* (PSM). Although the main idea of PSMs was briefly sketched previously [36], the formal development including proofs of soundness and completeness is first presented here. PSMs not only allow us to compactly represent (even infinite) sets of plans by a finite structure but mainly allow us to effectively implement operations crucial for our planning algorithms. Furthermore, we combine a previous extensibility approximation [17] with a method of a distributed relaxed heuristic used, for example, in MADLA planner [32]. This gives rise to a multiagent planner outperforming the state-of-the-art FMAP planner.

We use classical multiagent benchmark domains found in the literature to evaluate our planner. We provide a comprehensive domains description, and we compare experimental results with other state-of-the-art planners. Finally, we analyze the benchmark domains from the point of view of different privacy classifications. Different privacy classifications differ in facts explicitly revealed to other agents. While other planners are usually designed with a fixed privacy classification in mind, we show that our planner can be easily adjusted to work with various privacy classifications. Hence, we provide a user the freedom to choose what is public, and we can directly compare our planner with other planners. Furthermore, we show that a restricted public knowledge can even improve planner performance.

## 2 Related work

In classical planning, the plan synthesis process is typically a systematic search in either space of states, plans, or a combination of both. First multiagent planner for MA-STRIPS called planning first [28] used a global plan-space search and local forward state-space search. FMAP [34] is a representative of a multiagent partial ordered planner using heuristic search to locally improve efficiency and global distributed search in the space of the partial plans. A representative planners of the coordination-centric planning are distributed planning by graph merging (DPGM) [11,30], best response planning [21], and $\mu$-SATPLAN [10]. A planner Distoplan [12] and following A# planner [20] pioneered the idea of planning by means of finite-state machines (FSM). However, the A# planner was evaluated only on one planning domain. Our approach represents agent plans as FSMs as well. Additionally, we use a principle of intersection of the FSMs effectively acting as filter for unfeasible combination of plans of different agents. Moreover, our motivation was to provide a practical planning system; thus, we aimed at an efficient implementation and thorough experimental evaluation of our planner.

Distributed MA-STRIPS multiagent planners in the literature can be roughly separated to three groups by privacy preservation. Most of the planners follows concept of privacy by *information obfuscation* [4,27] or *information aggregation* [32,33]. With information obfuscation, agents are allowed to communicate private information with other agents as far as the information is obfuscated such that only the owning agent can understand it (e.g., the name of an action is replaced by a hash code). With information aggregation, the information is aggregated such that only the owning agent knows all details (e.g., a summed up cost of private actions can be send to other agents). An exception is the GPPP planner [26] providing *full privacy* by communicating only public information. Our approach also provides full privacy. Especially in the contrast to the obfuscation principle, we can reduce the size of plan space because privacy preservation act as natural abstraction of the problem from perspective of particular agents. Our principle thus allows lower requirements on the communicated data.

Multiagent planning can also be seen as a specific form of *factored planning* [1]. Factored planning tries to decompose a planning problem into possibly independent subproblems. Solving these subproblems scales linearly with the size of the domain and in the worst case exponentially with the size only of the largest subproblem and interactions among subproblems. Obviously, the catch is that not all planning problems can be factored enough to benefit from such efficiency gain. In [7], causal graphs [2] of the planning domains are used to identify when factorization is computationally beneficial. A practical algorithm based on this result and on principle of decomposition trees [9] was proposed in [22]. This principle can be also seen as a variation on localized planning [24]. The difference between multiagent planning and factored planning is that in multiagent planning the factorization is fixed and given by agent abilities.

## 3 Multiagent planning

Similarly as in classical planning, we assume a planning model based on extension of STRIPS [13] compactly representing a deterministic transition system. The multiagent extension follows the principles proposed in MA-STRIPS by Brafman and Domshlak in [6]. Agent capabilities are described as a finite repertoire of agent's STRIPS actions. The agent actions possibly affect only parts of the environment, thus inducing local planning problems of the particular agent. Aptly, this (partial) "separation of concerns" of the agents keeps the private information local. Also it helps to increase efficiency of the planning process by hiding parts irrelevant for other agents.

The agents are *cooperative* and *coordinated* and concurrently plan and execute their local plans in order to achieve a joint goal. The environment wherein the agents act is *classical* with *deterministic* actions. The following formal preliminaries restate the MA-STRIPS problem [6] and define *local planning problems* and plan *extensibility* [35] required for the following sections.

### 3.1 Planning problem

An MA-STRIPS *planning problem* $\Pi$ is a quadruple $\Pi = \langle P, \{\alpha_i\}_{i=1}^{n}, I, G\rangle$, where $P$ is a set of facts, $\alpha_i$ is the set of actions of $i$th agent, $I \subseteq P$ is an initial state, and $G \subseteq P$ is a set of goal facts. Selector functions $\mathsf{facts}(\Pi)$, $\mathsf{agents}(\Pi)$, $\mathsf{init}(\Pi)$, and $\mathsf{goal}(\Pi)$ are defined so that $\Pi = \langle \mathsf{facts}(\Pi), \mathsf{agents}(\Pi), \mathsf{init}(\Pi), \mathsf{goal}(\Pi)\rangle$ holds for any problem $\Pi$.

An *action* an agent can perform is a quadruple containing unique action id and three subsets of $\mathsf{facts}(\Pi)$ which in turn denote the set of *preconditions*, the set of *add effects*, and the set of *delete effects*. Action ids are arbitrary atomic objects, and we always consider ids to be unique within a given problem. Selector functions $\mathsf{id}(a)$, $\mathsf{pre}(a)$, $\mathsf{add}(a)$, and $\mathsf{del}(a)$ are defined so that $a = \langle \mathsf{id}(a), \mathsf{pre}(a), \mathsf{add}(a), \mathsf{del}(a)\rangle$ holds for any action $a$. Moreover, let $\mathsf{eff}(a) = \mathsf{add}(a) \cup \mathsf{del}(a)$.

An *agent* is identified with its capabilities; in other words, the $i$th agent $\alpha_i = \{a_1, \dots, a_m\}$ is determined by a finite set of actions it can preform in the environment. We use metavariable $\alpha$ to range over agents from $\Pi$. A *planning state* $s$ is a finite set of facts, and we say that fact $p$ *holds in* $s$, or that $p$ *is valid in* $s$, iff $p \in s$. When $\mathsf{pre}(a) \subseteq s$, *state progression* function $\gamma$ is defined classically as $\gamma(s, a) = (s \setminus \mathsf{del}(a)) \cup \mathsf{add}(a)$.

*Example 1* Throughout the paper, we shall use the following running example concerning a small logistic company. The company owns two transport vehicles (`plane` and `truck`) and operates three locations (`prague`, `brno`, and `ostrava`). A `plane` can travel from `prague` to `brno` and back, while a `truck` provides connection between `brno` and `ostrava`. The company receives a delivery job to transport the `crown` from `prague` to `ostrava`. The company manager needs to plan tasks for the vehicle operators so that the delivery job is done.

This delivery problem can be expressed using MA-STRIPS as follows. Actions `fly`($\mathit{loc}_1$, $\mathit{loc}_2$) and `drive`($\mathit{loc}_1$, $\mathit{loc}_2$) describe movement of `plane` and `truck`, respectively. Actions `load`(*veh*, *loc*) and `unload`(*veh*, *loc*) describe loading and unloading of `crown` by a given vehicle at a given location.

We define two agents *Plane* and *Truck*. The agents are defined by sets of executable actions as follows.

$$\begin{aligned}
\mathit{Plane} = \{\; &\texttt{fly(prague, brno), load(plane, prague), unload(plane, prague),}\\
&\texttt{fly(brno, prague), load(plane, brno), unload(plane, brno)}\;\}\\
\mathit{Truck} = \{\; &\texttt{drive(brno, ostrava), load(truck, brno), unload(truck, brno),}\\
&\texttt{drive(ostrava, brno), load(truck, ostrava), unload(truck, ostrava)}\;\}
\end{aligned}$$

Aforementioned actions are defined using facts $\texttt{at}(\mathit{veh}, \mathit{loc})$ to describe possible vehicle locations and facts $\texttt{in}(\texttt{crown}, \mathit{loc})$ and $\texttt{in}(\texttt{crown}, \mathit{veh})$ to describe positions of $\texttt{crown}$. We omit action ids in examples when no confusion can arise. For example, we have the following.

$$\begin{aligned}
\texttt{fly}(\mathit{loc}_1, \mathit{loc}_2) &= \langle\{\texttt{at}(\texttt{plane}, \mathit{loc}_1)\}, \{\texttt{at}(\texttt{plane}, \mathit{loc}_2)\}, \{\texttt{at}(\texttt{plane}, \mathit{loc}_1)\}\rangle\\
\texttt{load}(\mathit{veh}, \mathit{loc}) &= \langle\{\texttt{at}(\mathit{veh}, \mathit{loc}), \texttt{in}(\texttt{crown}, \mathit{loc})\}, \{\texttt{in}(\texttt{crown}, \mathit{veh})\}, \{\texttt{in}(\texttt{crown}, \mathit{loc})\}\rangle
\end{aligned}$$

The initial state and the goal are given as follows.

$$\begin{aligned}
I &= \{\texttt{at(plane, prague)}, \texttt{at(truck, brno)}, \texttt{in(crown, prague)}\}\\
G &= \{\texttt{in(crown, ostrava)}\}
\end{aligned}$$

The goal reflects the delivery requirement. □

### 3.2 Privacy classification of facts and actions

In MA-Strips multiagent planning, each fact is classified either as *public* or as *internal* out of computational or privacy concerns. MA-Strips specifies this classification as follows. A fact is *public* when it is mentioned by actions of at least two different agents. A fact is *internal for agent* $\alpha$ when it is not public but mentioned by some action of $\alpha$. A fact is *relevant for* $\alpha$ when it is either public or internal for $\alpha$. Relevant facts contain all the facts which agent $\alpha$ needs to understand, because other facts are internal for other agents and thus not directly concern $\alpha$. Formal definitions and notations used throughout the paper are presented in the upper parts of Fig. 1.

It is possible to extend the set of public facts to contain additionally some facts that would be internal by the above definition. This is important for our experimental evaluation because some multiagent planners use different facts classification. It is an advantage of our planner that it can be used with different facts classification because (1) we provide a user the freedom to choose what is public and (2) we can directly compare our planner with planners that use different classifications. The only requirement for our planner is that every fact shared by at least two agents is public. Furthermore, it is common in the literature [27] to require that all the goals are public. An MA-Strips problem with internal goals can be easily transformed to an equivalent problem without internal goals (see Sect. 7.3), and thus, we omit internal goals in formal presentation. Then $\mathsf{pub\text{-}facts}(\Pi)$ is defined as the minimal superset of the

| | | |
|---|---|---|
| $\mathsf{facts}(a)$ | $= \mathsf{pre}(a) \cup \mathsf{add}(a) \cup \mathsf{del}(a)$ | *facts of action a* |
| $\mathsf{facts}(\alpha)$ | $= \bigcup_{a\in\alpha} \mathsf{facts}(a)$ | *facts of agent* $\alpha$ |
| $\mathsf{pub\text{-}facts}(\Pi)$ | $= \bigcup_{\alpha\neq\beta}(\mathsf{facts}(\alpha) \cap \mathsf{facts}(\beta))$ | *public facts of* $\Pi(\alpha, \beta \in \mathsf{agents}(\Pi))$ |
| $\mathsf{int\text{-}facts}(\alpha)$ | $= \mathsf{facts}(\alpha) \setminus \mathsf{pub\text{-}facts}(\Pi)$ | *facts internal for agent* $\alpha$ |
| $\mathsf{rel\text{-}facts}(\alpha)$ | $= \mathsf{facts}(\alpha) \cup \mathsf{pub\text{-}facts}(\Pi)$ | *facts relevant for agent* $\alpha$ |
| $\mathsf{pub\text{-}actions}(\alpha)$ | $= \{a \in \alpha : \mathsf{eff}(a) \cap \mathsf{pub\text{-}facts}(\Pi) \neq \emptyset\}$ | *public actions of agent* $\alpha$ |
| $\mathsf{int\text{-}actions}(\alpha)$ | $= \alpha \setminus \mathsf{pub\text{-}actions}(\alpha)$ | *internal actions of agent* $\alpha$ |

**Fig. 1** MA-Strips privacy classification of facts and actions

intersection from the definition that satisfies $G \subseteq \textsf{pub-facts}(\Pi)$. In the rest of this paper, we suppose $G \subseteq \textsf{pub-facts}(\Pi)$ and also another simplification common in the literature [6] which says that $\alpha_i$ are pairwise disjoint.[1]

*Example 2* In our running example, the only fact shared by the two agents is `in(crown, brno)`. As we require $G \subseteq \textsf{pub-facts}(\Pi)$, we have the following facts classification.

$$\begin{aligned}\textsf{pub-facts}(\Pi) &= \{\ \texttt{in(crown, brno)}, \texttt{in(crown, ostrava)}\}\\ \textsf{int-facts}(\texttt{\textit{Plane}}) &= \{\ \texttt{at(plane, prague)}, \texttt{at(plane, brno)}, \texttt{in(crown, prague)}, \texttt{in(crown, plane)}\}\end{aligned}$$

□

MA-Strips further extends this classification of facts to actions as follows. An action is *public* when it has a public effect, otherwise it is *internal*. Strictly speaking, MA-Strips defines an action as public whenever it mentions a public fact even in a precondition (i.e., when $\textsf{facts}(a) \cap \textsf{pub-facts}(\Pi) \neq \emptyset$). However, our method of multiagent planning does not rely on synchronization of public preconditions, and hence, we can allow actions with only public preconditions to be internal. For our planner, it is enough to know that internal actions do not *modify* public state. Formal definitions and notations are presented in the lower part of Fig. 1.

## 3.3 Local planning problems

In MA-Strips problems, agent actions are supposed to manipulate a shared global state when executed. In multiagent planning with external actions, a *local planning problem* is constructed for every agent $\alpha$. Each local planning problem of $\alpha$ is a classical Strips problem containing $\alpha$'s own actions together with information about public actions of other agents. These local planning problems allow us to divide an MA-Strips problem to several Strips problems which can be solved separately by a classical planner. This paper describes a way how to find a solution of an MA-Strips problem, but it does not address the question of *execution* of a plan in some real-world environment.

The *projection* $F \triangleright \alpha$ of set of facts $F$ to agent $\alpha$ is the restriction of $F$ to the facts relevant for $\alpha$. Hence, projection removes from $F$ facts not relevant for $\alpha$, and thus, it represents $F$ as understood by agent $\alpha$. The *projection* $a \triangleright \alpha$ of action $a$ to agent $\alpha$ removes from $a$ facts not relevant for $\alpha$, again representing $a$ as seen by $\alpha$. The projections are formally defined as follows.

**Definition 1** Given $\Pi$, let $F$ be an arbitrary set $F \subseteq \textsf{facts}(\Pi)$ of facts and let $a$ be an action from $\Pi$. The *projection $F \triangleright \alpha$ of $F$ to $\alpha \in$* $\textsf{agents}(\Pi)$ and the *projection $a \triangleright \alpha$ of action $a$ to $\alpha$* are defined as follows.

$$F \triangleright \alpha = F \cap \textsf{rel-facts}(\alpha) \qquad a \triangleright \alpha = \langle \textsf{id}(a), \textsf{pre}(a) \triangleright \alpha,\ \textsf{add}(a) \triangleright \alpha,\ \textsf{del}(a) \triangleright \alpha \rangle$$

The action projection is extended to sets of actions element-wise. □

Note that $a \triangleright \alpha = a$ when $a \in \alpha$. Hence, projection to $\alpha$ alters only actions of agents other than $\alpha$. Also note that action ids are preserved under projection.

[1] This rules out *joint actions*. Any MA-Strips problem with joint actions can be translated to an equivalent problem without joint actions. However, a solution that would take advantage joint actions is left for future research.

*Example 3* In our example, we have the following.

$$\begin{aligned}\texttt{fly(prague, brno)} \triangleright \textit{Plane} &= \texttt{fly(prague, brno)}\\ \texttt{fly(prague, brno)} \triangleright \textit{Truck} &= \langle\emptyset, \emptyset, \emptyset\rangle\\ \texttt{load(truck, brno)} \triangleright \textit{Plane} &= \langle\{\texttt{in(crown, brno)}\}, \emptyset, \{\texttt{in(crown, brno)}\}\rangle\\ \texttt{unload(truck, ostrava)} \triangleright \textit{Plane} &= \langle\emptyset, \{\texttt{in(crown, ostrava)}\}, \emptyset\rangle\end{aligned}$$

□

In multiagent planning with *external actions*, every agent $\alpha$ is from the beginning equipped with projections of public actions of other agents. These projections, which we call external actions, describe how agent $\alpha$ sees effects of public actions of other agents. In a local planning problem, an agent needs external actions so that he can create a plan which contains also public actions of other agents. The set of actions in a local planning problem of agent $\alpha$ simply contains actions of agent $\alpha$ together with external actions of $\alpha$. Now it is easy to define a *local planning problem* $\Pi \triangleright \alpha$ of agent $\alpha$ also called *projection of* $\Pi$ to $\alpha$ as a classical STRIPS problem. The set of facts $P$ and the initial state $I$ are restricted to those facts relevant for $\alpha$. There is no need to restrict the goal $G$ because all the goal facts are public and thus relevant for all the agents. A formal definition follows.

**Definition 2** Given an MA-STRIPS problem $\Pi$, the *local planning problem* $\Pi \triangleright \alpha$ of agent $\alpha$ is defined for every $\alpha \in \mathsf{agents}(\Pi)$ as the classical STRIPS problem

$$\Pi \triangleright \alpha = \langle \mathsf{facts}(\Pi) \triangleright \alpha,\ \alpha \cup \mathsf{ext\text{-}actions}(\alpha),\ \mathsf{init}(\Pi) \triangleright \alpha,\ G\rangle$$

where the set $\mathsf{ext\text{-}actions}(\alpha)$ of *external actions of agent* $\alpha$ is defined as follows.

$$\mathsf{ext\text{-}actions}(\alpha) = \bigcup_{\beta \neq \alpha}(\mathsf{pub\text{-}actions}(\beta) \triangleright \alpha) \qquad \text{(for all } \beta \in \mathsf{agents}(\Pi))$$

□

*Example 4* In our example, all the actions arranging vehicle movements are internal. Public are only the actions providing package treatment at public locations (`brno`, `ostrava`). Hence, the set $\mathsf{pub\text{-}actions}(\textit{Plane})$ contains only actions `load(plane, brno)` and `unload(plane, brno)`, while $\mathsf{pub\text{-}actions}(\textit{Truck})$ is as follows.

{ `load(truck, brno)`, `unload(truck, brno)`, `load(truck, ostrava)`, `unload(truck, ostrava)` }

Hence, $\mathsf{ext\text{-}actions}(\textit{Truck})$ has two actions and $\mathsf{ext\text{-}actions}(\textit{Plane})$ has four actions. This yields the local problem $\Pi \triangleright \textit{Plane}$ with 10 actions and the problem $\Pi \triangleright \textit{Truck}$ with eight actions. □

## 3.4 Planning with external actions

We would like to solve agent local problems separately and compose local solutions to a global solution of $\Pi$. However, not all local solutions can be easily composed to a solution of $\Pi$. Concepts of *public plans* and their *extensibility* help us to recognize local solutions which are conductive to this aim.

A *plan* $\pi$ is a sequence of actions $\langle a_1, \ldots, a_k\rangle$. A plan $\pi$ defines an order in which the actions are executed by their unique owner agents. It is supposed that independent actions can be executed in parallel. A *solution* of $\Pi$ is a plan $\pi$ whose execution transforms the initial state $I$ to the state $s$ such that $G \subseteq s$. A *local solution* of agent $\alpha$ is a solution of the local planning problem $\Pi \triangleright \alpha$. Let $\mathsf{sols}(\Pi)$ and $\mathsf{sols}(\Pi \triangleright \alpha)$ denote the sets of all the solutions of MA-STRIPS problem $\Pi$ and all the local solutions of $\alpha$, respectively.

*Example 5* Let us consider the following plans.

$$\begin{aligned}
\pi_0 = \langle\ & \texttt{load(plane, prague)},\ \texttt{fly(prague, brno)},\ \texttt{unload(plane, brno)},\\
& \texttt{load(truck, brno)},\ \texttt{drive(brno, ostrava)},\ \texttt{unload(truck, ostrava)}\ \rangle\\
\pi_1 = \langle\ & \texttt{unload(truck, ostrava)} \triangleright \mathtt{Plane}\ \rangle\\
\pi_2 = \langle\ & \texttt{unload(plane, brno)} \triangleright \mathtt{Truck},\ \texttt{load(truck, brno)},\\
& \texttt{drive(brno, ostrava)},\ \texttt{unload(truck, ostrava)}\ \rangle
\end{aligned}$$

It is easy to check that $\pi_0$ is a solution of our example MA-STRIPS problem $\Pi$. Plan $\pi_1$ is a solution of $\Pi \triangleright \mathtt{Plane}$ because projection $\texttt{unload(truck, ostrava)} \triangleright \mathtt{Plane}$ of $\mathtt{Truck}$'s public action simply produces the goal state out of the blue. Finally, $\pi_2 \in \mathsf{sols}(\Pi \triangleright \mathtt{Truck})$. □

A *public plan* $\sigma$ is a plan that contains only public actions. A public plan can be seen as a solution outline that captures execution order of public actions while ignoring agents internal actions. A public plan can be safely sent to any agent because it contains only public information. In order to avoid confusions between public and external versions of the same action, we formally define public plans to contain only public action *id*s. For a plan $\pi$ of $\Pi$ (or a plan of $\Pi \triangleright \alpha$), we define the *public projection* $\pi \triangleright \star$ of $\pi$ as the sequence of all public action *id*s from $\pi$ preserving their order. Public projection of a plan thus removes any internal actions from $\pi$. Formal definition follows.

**Definition 3** A *public plan* $\sigma$ is a sequence of public action *id*s. Given a plan $\pi$ of $\Pi$ (or of $\Pi \triangleright \alpha$), the public projection $\pi \triangleright \star$ of $\pi$ is defined to be the public plan $\pi \triangleright \star = \langle \mathsf{id}(a) : a \in \pi \text{ and } a \in \mathsf{pub\text{-}actions}(\Pi)\rangle$. Public projection is extended to sets of plans element-wise. □

*Example 6* In our example, we know that $\pi_0 \in \mathsf{sols}(\Pi)$ and $\pi_1 \in \mathsf{sols}(\Pi \triangleright \mathtt{Plane})$ and $\pi_2 \in \mathsf{sols}(\Pi \triangleright \mathtt{Truck})$. Thus, we can construct the following public solutions.

$$\begin{aligned}
\pi_0 \triangleright \star &= \langle\ \mathsf{id}(\texttt{unload(plane, brno)}), \mathsf{id}(\texttt{load(truck, brno)}), \mathsf{id}(\texttt{unload(truck, ostrava)})\rangle\\
\pi_1 \triangleright \star &= \langle\ \mathsf{id}(\texttt{unload(truck, ostrava)})\rangle\\
\pi_2 \triangleright \star &= \langle\ \mathsf{id}(\texttt{unload(plane, brno)}), \mathsf{id}(\texttt{load(truck, brno)}), \mathsf{id}(\texttt{unload(truck, ostrava)})\rangle
\end{aligned}$$

Note that $\pi_0 \triangleright \star = \pi_2 \triangleright \star$ and also note that we have omitted the projection operator ($\triangleright$) because ids are preserved under projection. □

From every solution $\pi$ of $\Pi$ (or of $\Pi \triangleright \alpha$), we can construct a uniquely determined public plan $\sigma = \pi \triangleright \star$. On the other hand, for a single public plan $\sigma$ there might be more than one, or none, solutions with public projection $\sigma$. A public plan $\sigma$ is called *extensible* when there is a solution of $\Pi$ with public projection $\sigma$. Similarly, when there is a solution of $\Pi \triangleright \alpha$ with public projection $\sigma$, then $\sigma$ is called *$\alpha$-extensible*. Extensible public plans give us an order of public actions which is acceptable for all the agents. Thus, extensible public plans are very close to solutions of $\Pi$, and it is relatively easy to construct a solution of $\Pi$ once we have an extensible public plan. Hence, our algorithms will aim at finding extensible public plans. The following formally defines public plan extensibility.

**Definition 4** Let $\sigma$ be a public plan of $\Pi$.

$$\begin{aligned}
\sigma \text{ is } \mathit{extensible} \quad & \text{iff } \exists \pi \in \mathsf{sols}(\Pi) : \pi \triangleright \star = \sigma\\
\sigma \text{ is } \alpha\text{-}\mathit{extensible} \quad & \text{iff } \exists \pi \in \mathsf{sols}(\Pi \triangleright \alpha) : \pi \triangleright \star = \sigma
\end{aligned}$$

□

*Example 7* In our example, we can see that $\pi_0 \rhd \star$ is extensible because it was constructed from the solution of $\Pi$. For the same reason, we see that $\pi_1 \rhd \star$ is $\mathtt{Plane}$-extensible and $\pi_2 \rhd \star$ is $\mathtt{Truck}$-extensible. It is easy to see that $\pi_2 \rhd \star$ is also $\mathtt{Plane}$-extensible. However, $\pi_1 \rhd \star$ is not $\mathtt{Truck}$-extensible because $\mathtt{Truck}$ needs to execute other public actions prior to $\mathtt{unload(truck, ostrava)}$. □

The following proposition states the correctness of the multiagent planning with external actions. It establishes the relationship between extensible and $\alpha$-extensible plans. Its direct consequence is that to find a solution of $\Pi$ it is enough to find a local solution $\pi_\alpha \in \mathsf{sols}(\Pi \rhd \alpha)$ which is $\beta$-extensible for every other agent $\beta$. A constructive proof follows.

**Theorem 1** ([35]) *Public plan $\sigma$ of $\Pi$ is extensible if and only if $\sigma$ is $\alpha$-extensible for every agent $\alpha \in \mathsf{agents}(\Pi)$.*

*Example 8* We have seen previously that $\pi_2 \rhd \star$ is $\mathtt{Truck}$-extensible and also $\mathtt{Plane}$-extensible. Hence, we know that there is some solution of $\Pi$ even without knowing $\pi_0$. Furthermore, the proof of Theorem 1 shows how to reconstruct the solution. On the other hand, we know that $\pi_1 \rhd \star$ is not $\mathtt{Truck}$-extensible and thus $\pi_1 \rhd \star$ is not extensible. □

## 4 Planning state machines (PSM)

The basic idea behind the multiagent planning algorithm described in this paper is based on Theorem 1, and it can be described briefly as follows. Every agent $\alpha$ keeps generating new solutions of its local planning problem $\Pi \rhd \alpha$ and announces their public projections to all the other agents. Hence, the set $\Delta_\alpha$ of public plans generated so far by $\alpha$ is known by all the agents. Once there is a single public plan $\sigma$ generated by all the agents, we can stop the algorithm yielding $\sigma$ as the public solution of $\Pi$. This is because every plan generated by agent $\beta$ is automatically $\beta$-extensible and hence $\sigma$ is extensible by Theorem 1.

In this section, we utilize finite-state machines to effectively represent sets of plans (or public plans) of a STRIPS problem mentioned in the above algorithm description. These finite-state machines, which we call *planning state machines* (PSM), are described in Sect. 4.1. PSMs allow us to effectively implement operations which are crucial for our multiagent planning algorithm. These operations are (1) adding a new solution to an existing PSM (Sect. 4.2), (2) computing a public projection of a PSM (Sect. 4.3), and (3) intersecting public projections of PSMs (Sect. 4.4).

### 4.1 Basics of planning state machines

Finite-state machines [16] are widely used in computer science for manifold purposes. In this section, we utilize state machines to recognize and compute solutions of STRIPS and MA-STRIPS planning problems. To achieve this, we use the set of planning actions $A$ as an alphabet while planning states (sets of facts) become states of our *planning state machine* (PSM). PSM state-transition $\delta$ simply resembles planning state progression function $\gamma$. Hence, a PSM accepts words over $A$, that is, plans.

For our purposes, a deterministic finite-state machine (DFS) is a tuple $\langle \Sigma, S, s_0, \delta, F \rangle$ where $\Sigma$ is a finite alphabet, $S$ is a finite set of states, $s_0 \in S$ is an initial state, $\delta$ is a complete state-transition function ($\delta : S \times \Sigma \to S$), and $F \subseteq S$ is a set of accepting states. A non-deterministic finite-state machine (NFS) is a tuple $\langle \Sigma, S, s_0, \delta, F \rangle$ much like a DFS, but the state-transition function is non-deterministic, that is, $\delta : S \times \Sigma \to \mathcal{P}(S)$. For a DFS (for an NFS respectively), the state-transition function $\delta$ can be naturally extended to $\delta^\star : S \times \Sigma^\star \to S$ (respectively to $\delta^\star : S \times \Sigma^\star \to \mathcal{P}(S)$) where $\Sigma^\star$ is the set of all finite words over alphabet $\Sigma$. A word $w \in \Sigma^\star$ is *accepted* by a DFS when $\delta^\star(s_0, w) \in F$. A word $w \in \Sigma^\star$ is *accepted* by an NFS when $\delta^\star(s_0, w) \cap F \neq \emptyset$.

**Definition 5** A *planning state machine (PSM)* of a STRIPS problem $\Pi = \langle P, A, I, G\rangle$ is a DFS $\Gamma = \langle \Sigma, S, I, \delta, F\rangle$ where

(1) alphabet $\Sigma$ is the set action ids $\Sigma = \{\mathsf{id}(a) : a \in A\}$,
(2) states are sets of facts ($S \subseteq \mathcal{P}(P)$) with $I \in S$,
(3) transitions satisfy that $\delta(s, \mathsf{id}(a)) = s'$ implies $\gamma(s, a) = s'$,
(4) and accepting states are $F = \{s \in S : G \subseteq s\}$.

Let $\mathsf{accept}(\Gamma)$ denote the set of all plans accepted by $\Gamma$. □

In general, a PSM does not need to contain all possible planning states of $\Pi$ because $S$ is only required to be a subset of $\mathcal{P}(P)$ which contains $I$. However, the following *soundness* result can be trivially proved for any PSM.

**Lemma 1** *Let $\Gamma$ be a PSM of a* STRIPS *problem $\Pi$. Then* $\mathsf{accept}(\Gamma) \subseteq \mathsf{sols}(\Pi)$.

*Proof* Follows directly from Definition 5 and definition of $\gamma$. □

The opposite inclusion does not necessarily hold. However, for a given STRIPS problem $\Pi$ we can easily construct a *complete* PSM $\Gamma$ which accepts all the solutions of $\Pi$. A complete PSM needs to contain all the possible transitions and all (reachable) planning states. A complete PSM can be constructed by a breadth-first search starting from the initial state and by adding all reachable planning states together with all possible transitions. Of course, this construction is highly ineffective, but it shall be used below to demonstrate basic operations on PSMs. The following defines a complete PSM.

**Definition 6** A PSM $\Gamma = \langle \Sigma, S, I, \delta, F\rangle$ of $\Pi = \langle P, A, I, G\rangle$ is *complete* when

(1) $S = \mathcal{P}(P)$, and
(2) transitions additionally satisfy that $\gamma(s, a) = s'$ implies $\delta(s, \mathsf{id}(a)) = s'$ whenever $\gamma(s, a)$ is defined. □

Hence, a complete PSM accepts every solution of $\Pi$, formally as follows.

**Lemma 2** *For a complete PSM $\Gamma$ of $\Pi$, it holds that* $\mathsf{accept}(\Gamma) = \mathsf{sols}(\Pi)$.

*Proof* Follows directly from Lemma 1 and the definition of $\gamma$. □

### 4.2 Extending a PSM with solutions

The first operation we define on PSMs is extending an existing PSM with a new solution $\pi$. The operation is denoted $\Gamma \oplus \pi$, and its result is an extended PSM which accepts all the plans as $\Gamma$ and additionally $\pi$. The operation is implemented simply by traversing $\pi$ and by adding corresponding states and transitions to $\Gamma$. The following constructive definition suggests an implementation linear in the size of the added solution. Note that in the definition we consider $\delta$ to be a set of triples, writing $\langle s, a, s'\rangle \in \delta$ instead of $s' \in \delta(s, a)$.

**Definition 7** Let a PSM $\Gamma = \langle \Sigma, S, I, \delta, F\rangle$ of $\Pi$ and a solution $\pi = \langle a_1, \ldots, a_n\rangle$ of $\Pi$ be given. Denote $s_0 = I$ and $s_i = \gamma(s_{i-1}, a_{i-1})$ for $0 < i \leq n$. The PSM $\Gamma \oplus \pi$ is defined as follows.

$$\Gamma \oplus \pi = \langle \Sigma, S \cup \{s_0, \ldots, s_n\}, I, \delta \cup \{\langle s_{i-1}, \mathsf{id}(a_{i-1}), s_i\rangle : 0 < i \leq n\}, F \cup \{s_n\}\rangle$$

□

The operation $\oplus$ can extend the set of accepted plans by more than $\pi$. However, the following lemma states that the PSM $\Gamma \oplus \pi$ accepts all the plans as $\Gamma$, and additionally other plans including $\pi$. Additionally accepted plans other than $\pi$ do not cause any problem because Lemma 1 ensures that every additionally accepted plan is a solution of $\Pi$. Important is that $\mathsf{accept}(\Gamma \oplus \pi) \subseteq \mathsf{sols}(\Pi)$ holds.

**Lemma 3** *Let $\Pi$ be a classical* STRIPS *problem, let $\Gamma$ be a PSM of $\Pi$, and let $\pi \in \mathsf{sols}(\Pi)$. Then $\Gamma \oplus \pi$ is correctly defined and* $\mathsf{accept}(\Gamma) \cup \{\pi\} \subseteq \mathsf{accept}(\Gamma \oplus \pi)$.

*Proof* Follows from Definition 7. □

## 4.3 Public planning state machines

Previous sections define a planning state machine $\Gamma$ which represents the set $\mathsf{accept}(\Gamma)$ of plans. From $\Gamma$, we would like to compute the corresponding set of public plans, that is, the set $\mathsf{accept}(\Gamma) \rhd \star = \{\pi \rhd \star : \pi \in \mathsf{accept}(\Gamma)\}$. In this section, we achieve this by transforming PSM $\Gamma$ to a *public planning state machine* which (1) accepts exactly the aforementioned set of public plans and (2) contains only public information. We call this operation the *public projection of PSM $\Gamma$*, and we denote it $\Gamma \rhd \star$.

Public PSMs will be exchanged among agents during our multiagent planning algorithm. Therefore, out of privacy concerns, it is essential that public PSMs contain only public information. A first attempt to construct a public PSM from PSM $\Gamma$ would be to treat internal actions as $\varepsilon$-transitions and eliminate them from $\Gamma$ using standard algorithm. The standard algorithm to eliminate $\varepsilon$-closures simply "bridges" $\varepsilon$-transitions with new transitions. As for internal facts contained within states, a first attempt is simply to delete them. Let us consider the example PSM $\Gamma_1$ from Fig. 2 (left). After eliminating internal transitions and

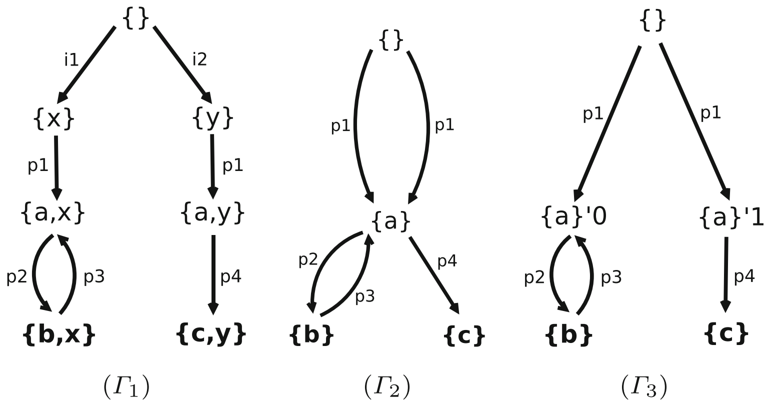

**Fig. 2** A motivation example for computing PSM public projection. We suppose a context where $\mathsf{p}n$ are public and $\mathsf{i}n$ internal actions and where a, b, c are public and x, y internal facts. Accepting states are marked bold. The initial state is {}

after deleting internal facts from states, we obtain the PSM $\Gamma_2$ (Fig. 2, middle). Unfortunately, $\Gamma_2$ also accepts the plan $\langle \mathsf{p1}, \mathsf{p2}, \mathsf{p3}, \mathsf{p4} \rangle$ which is not a public projection of any plan accepted by $\Gamma_1$. The problem is that two different states of $\Gamma_1$, namely $\{\mathsf{a}, \mathsf{x}\}$ and $\{\mathsf{a}, \mathsf{y}\}$, were merged in $\Gamma_2$ after removing internal facts $\mathsf{x}$ and $\mathsf{y}$. To solve this problem, we introduce integer marks to distinguish states which would otherwise became equal after removing internal facts. This is demonstrated by PSM $\Gamma_3$ (Fig. 2, right). It is easy to check that $\Gamma_3$ accepts exactly public projections of the plans accepted by $\Gamma_1$. Also note that $\Gamma_3$ is non-deterministic because of the non-deterministic transitions from the initial state. Hence, public projection can introduce non-determinism.

In order to formally define public PSMs, we need to define *public projections of states* and *actions*. The public projection $F \triangleright \star$ of a set of facts $F$ is simply the restriction of $F$ to public facts. The public projection $a \triangleright \star$ of action $a$ restricts facts in $a$ to public facts preserving action id.

**Definition 8** Let $\Pi$ be an MA-Strips problem. Let $F$ be an arbitrary set $F \subseteq \mathsf{facts}(\Pi)$ and let $a$ be an action from $\Pi$. The *public projection* $F \triangleright \star$ *of* $F$ and the *public projection* $a \triangleright \star$ *of* action $a$ are defined as follows.

$$F \triangleright \star = F \cap \mathsf{pub\text{-}facts}(\Pi) \qquad a \triangleright \star = \langle \mathsf{id}(a), \mathsf{pre}(a) \triangleright \star,\ \mathsf{add}(a) \triangleright \star,\ \mathsf{del}(a) \triangleright \star \rangle$$

Public projection is extended to sets of actions element-wise. □

The previous discussion explained why public PSMs need to contain integer-labeled states and why public PSMs need to be non-deterministic. Hence a public PSM of an MA-Strips problem $\Pi$ is an NFS with the following properties.

**Definition 9** A *public PSM* of an MA-Strips problem $\Pi$ is an NFS $\Delta = \langle \Sigma, S, I_0, \delta, F \rangle$ where

(1) the alphabet is $\Sigma = \{\mathsf{id}(a) : a \in \mathsf{pub\text{-}actions}(\Pi)\}$,
(2) states are integer-labeled sets of public facts ($S \subseteq \mathcal{P}(\mathsf{pub\text{-}facts}(\Pi)) \times \mathbb{N}$),
(3) the initial state is $I_0 = \langle \mathsf{init}(\Pi) \triangleright \star, 0 \rangle$ and $I_0 \in S$,
(4) transitions satisfy that $\langle s', i' \rangle \in \delta(\langle s, i \rangle, \mathsf{id}(a))$ implies $\gamma(s, a \triangleright \star) = s'$,
(5) and $\langle s, i \rangle \in F$ implies $\mathsf{goal}(\Pi) \subseteq s$.

Let $\mathsf{accept}(\Delta)$ denote the set of all plans accepted by $\Delta$. □

Now we describe the public projection algorithm to compute $\Gamma \triangleright \star$ from $\Gamma$. It is motivated by the standard $\varepsilon$-elimination algorithm [16, Chapter 2.5] extended with integer-mark introduction and public projection of states. For every state $s$, the *internal closure set* $\mathsf{int\text{-}closure}_\Gamma(s)$ contains all the states reachable from $s$ by internal transitions only. The set $\mathsf{int\text{-}closure}_\Gamma(s)$ can be computed by a DFS, and the following definition gives its semantics. We omit the index $\Gamma$ when no confusion can arise.

**Definition 10** Given PSM $\Gamma$ of agent local problem $\Pi \triangleright \alpha$, an *internal closure* of $s_0$, denoted $\mathsf{int\text{-}closure}(s_0)$, is the least set of states such that

(1) $s_0 \in \mathsf{int\text{-}closure}(s_0)$, and
(2) whenever $s \in \mathsf{int\text{-}closure}(s_0)$ for some $s$ then for all $a \in \mathsf{int\text{-}actions}(\alpha)$ it holds that $\delta(s, \mathsf{id}(a)) \in \mathsf{int\text{-}closure}(s_0)$.

In other words, the set $\mathsf{int\text{-}closure}(s_0)$ contains $s_0$ and all the states reachable from $s_0$ by transitions corresponding to internal actions. □

**Algorithm 1:** Algorithm to compute the public projection $\Gamma \triangleright \star$ of PSM $\Gamma$.

```
1  Function PublicProjection($\Gamma$) is
2      $\langle \Sigma, S, I, \delta, F \rangle \leftarrow \Gamma$;
3      $\Sigma_0 \leftarrow \{\mathsf{id}(a) : a \in \mathsf{pub\text{-}actions}(\Pi)\}$;
4      $I_0 \leftarrow \langle I \triangleright \star, 0 \rangle$;
5      $S_0 \leftarrow \{I_0\}$;
6      $\rho \leftarrow \emptyset$;    // initialize state renaming, $\rho : S \rightarrow \mathcal{P}(\mathsf{pub\text{-}facts}(\Pi)) \times \mathbb{N}$
7      $\rho(I) \leftarrow I_0$;    // set value of $\rho(I)$ to $I_0$
8      foreach $s \in (S \setminus \{I\})$ do
9          $\rho(s) \leftarrow \langle s \triangleright \star, |S_0| \rangle$;    // $|S_0|$ increases with every iteration
10         $S_0 \leftarrow S_0 \cup \rho(s)$;
11     end
12     $\delta_0 \leftarrow \emptyset$;    // initialize new transitions, $\delta_0 : S_0 \times \Sigma_0 \rightarrow \mathcal{P}(S_0)$
13     $F_0 \leftarrow \emptyset$;
14     foreach $s \in S$ do    // for every original state of $\Gamma$
15         $\{r_1, \ldots, r_k\} \leftarrow \mathsf{int\text{-}closure}(s)$;
16         foreach $id \in \Sigma_0$ do
17             $\delta_0(\rho(s), id) \leftarrow \{\rho(\delta(r_i, id)) : 0 < i \leq k\}$;
18         end
19         if $\mathsf{int\text{-}closure}(s) \cap F \neq \emptyset$ then
20             $F_0 \leftarrow F_0 \cup \{\rho(s)\}$;    // mark $\rho(s)$ as an accepting state
21         end
22     end
23     $\Delta \leftarrow \langle \Sigma_0, S_0, I_0, \delta_0, F_0 \rangle$;
24     return $\Delta$;
25 end
```

Once internal closures are computed for every state of $\Gamma$, the public projection algorithm proceeds as described by Algorithm 1. The role of the state renaming $\rho$ is to translate states of $\Gamma$ to states of the public projection. For every state $s$ of $\Gamma$, the renaming defines the state $\rho(s)$ in the constructed public PSM consisting of the public projection of $s$ and a unique integer mark. The second foreach cycle which starts at line 14 takes care of "bridging" of internal transitions. When state $s'$ is reachable from $s$ in $\Gamma$ by (zero or more) internal transitions followed by one public transition $\mathsf{id}(a)$, $\Gamma \triangleright \star$ will contain a transition from $\rho(s)$ to $\rho(s')$ labeled with $\mathsf{id}(a)$. Finally, the condition at line 19 marks accepting states.

The PSM public projection algorithm can be alternatively explained on the example from Fig. 3. PSM $\Gamma_1$ (Fig. 3, left) is the input PSM. PSM $\Gamma_2$ (Fig. 3, middle) is obtained from $\Gamma_1$ by eliminating internal transitions. PSM $\Gamma_3$ (Fig. 3, right) is obtained from $\Gamma_2$ by public projection of states and by marks introduction. Note that $\Gamma_3$ additionally compresses $\Gamma_2$ by unifying states with equal public projection which has equal sets of outgoing transitions. This is an optimization implemented in our planner but omitted from formal presentation. Another optimization is to remove states unreachable from the initial state and to remove states from which no accepting state is reachable. None of the above optimizations affects the semantics.

The following definition defines $\Gamma \triangleright \star$ as the result of Algorithm 1 and formally states algorithm correctness.

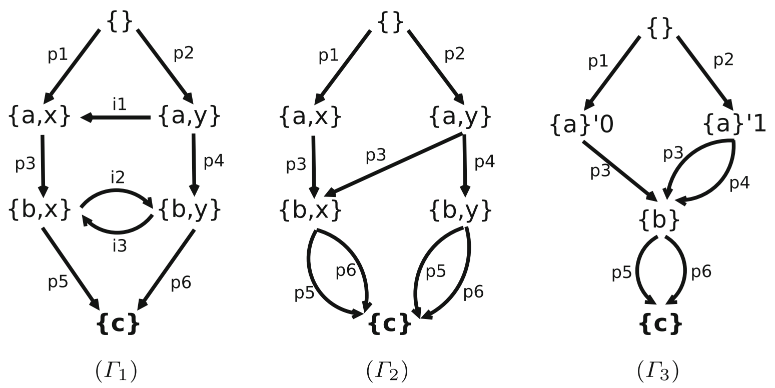

**Fig. 3** Example of computing PSM public projection. We suppose a context where p$n$ are public and i$n$ internal actions and where a, b, c are public and x, y internal facts. Accepting states are marked *bold*. The initial state is {}

**Definition 11** Let $\Pi \triangleright \alpha$ be a local problem of agent $\alpha$ and let $\Gamma$ be a PSM of $\Pi \triangleright \alpha$. The *public projection* of $\Gamma$, denoted $\Gamma \triangleright \star$, is the result of Algorithm 1. □

**Lemma 4** *Let $\Pi \triangleright \alpha$ be a local problem of agent $\alpha$ and let $\Gamma$ be a PSM of $\Pi \triangleright \alpha$. Then $\Gamma \triangleright \star$ is a public PSM of $\Pi$ and* $\mathsf{accept}(\Gamma \triangleright \star) = \mathsf{accept}(\Gamma) \triangleright \star$.

*Proof* The inclusion ($\subseteq$) is proved by extending a public solution $\sigma \in \mathsf{accept}(\Gamma \triangleright \star)$ with internal actions to a solution $\pi \in \mathsf{accept}(\Gamma)$. The opposite inclusion ($\supseteq$) is proved by simulating a plan $\pi \in \mathsf{accept}(\Gamma)$ in the state space of $\Gamma$ and by constructing a corresponding simulation of $\pi \triangleright \star$ in the state space of $\Gamma \triangleright \star$. □

## 4.4 Intersection of public PSMs

The previous section describes how to compute the public projection of a PSM. Suppose we have two public PSMs of an MA-STRIPS problem $\Pi$. This section describes how to compute an intersection of two public PSMs which is a public PSM which accepts the plans accepted by both the original PSMs.

There is a standard algorithm [16, Theorem 4.8] to compute an intersection of two arbitrary NFSs. The standard algorithm defines an intersection of NFS $\Delta_1$ and NSF $\Delta_2$ as a new NFS whose set of states is the Cartesian product of states of $\Delta_1$ and $\Delta_2$. The intersection of NFSs contains a transition between two states when there are corresponding transitions in both the original NFSs $\Delta_1$ and $\Delta_2$. This standard algorithm, however, needs to be adjusted because the standard algorithm applied to public PSMs would not yield a correctly defined public PSM. The reason is that the structure of states in a public PSM is fixed.

We want to compute an intersection of two public PSMs (of the same MA-STRIPS problem $\Pi$). We can take advantage of the fact that both public PSMs are defined on the same set of states. Moreover, a transition from state $\langle s, i\rangle$ labeled by action $a$ uniquely determines $s'$ in the destination state $\langle s', i'\rangle$. Hence, we do not need to define the set of states in an intersection as a Cartesian product, but we can use integer-labeled public states and only adjust integer marks appropriately. Thus, an intersection of two public PSMs will be a public PSM. To combine integer marks, we can use arbitrary but fixed injective function from $\mathbb{N} \times \mathbb{N}$ to $\mathbb{N}$ such that $0 \cdot 0 = 0$. A classical example is the Cantor pairing function.[2]

[2] $i \cdot j = \frac{(i+j)(i+j+1)}{2} + j$.

The following defines intersection of public PSMs $\Delta_1$ and $\Delta_2$ and proves its correctness. The set of states of an intersection PSM is constructed using the Cantor pairing function as follows. Whenever there is a state $\langle s, i\rangle$ in $\Delta_1$ and also a state $\langle s, j\rangle$ in $\Delta_2$, then the intersection PSM contains the state $\langle s, i \cdot j\rangle$. Hence, every state of the intersection PSM corresponds to uniquely determined states in $\Delta_1$ and $\Delta_2$. The state transition function of an intersection PSM emulates transition functions of both input public PSMs. A state in the intersection PSM is accepting when both the corresponding states are accepting in $\Delta_1$ and $\Delta_2$. Finally, note that the Cantor pairing function is not commutative and thus the intersection operation is commutative up to the integer marks. Nevertheless, all possible resulting public PSMs are equal with respect to the set of accepted plans.

**Definition 12** Let $\Delta_1 = \langle \Sigma, S_1, I, \delta_1, F_1\rangle$ and $\Delta_2 = \langle \Sigma, S_2, I, \delta_2, F_2\rangle$ be two public PSMs of an MA-STRIPS problem $\Pi$. Let $\cdot$ be the Cantor pairing function. The *intersection* of $\Delta_1$ and $\Delta_2$ is a public PSM $\Delta_0 = \langle \Sigma, S_0, I, \delta_0, F_0\rangle$ of problem $\Pi$ where

(1) $S_0 = \{\langle s, i \cdot j\rangle : \langle s, i\rangle \in S_1$ and $\langle s, j\rangle \in S_2\}$, and
(2) $\langle s', i' \cdot j'\rangle \in \delta_0(\langle s, i \cdot j\rangle, id)$ iff $\langle s', i'\rangle \in \delta_1(\langle s, i\rangle, id)$ and $\langle s', j'\rangle \in \delta_2(\langle s, j\rangle, id)$,
(3) and $F_0 = \{\langle s, i \cdot j\rangle : \langle s, i\rangle \in F_1$ and $\langle s, j\rangle \in F_2\}$.

The intersection of $\Delta_1$ and $\Delta_2$ is denoted $\Delta_1 \cap \Delta_2$. □

**Lemma 5** *The intersection $\Delta_1 \cap \Delta_2$ of two public PSMs $\Delta_1$ and $\Delta_2$ of $\Pi$ is a correctly defined public PSM of $\Pi$ and the following holds.*

$$\mathsf{accept}(\Delta_1 \cap \Delta_2) = \mathsf{accept}(\Delta_1) \cap \mathsf{accept}(\Delta_2)$$

*Proof* Follows from Definitions 9 and 12. □

## 4.5 Multiagent planning with complete PSMs

The results from the previous sections now make it easy to introduce Algorithm 2 to compute all public solutions of a given MA-STRIPS problem $\Pi$. For every agent $\alpha$, the algorithm computes the complete PSM of $\Pi \triangleright \alpha$ and its public projection. All the public PSMs are then intersected, and their intersection is returned as a result. Theorem 2 states that the intersection contains exactly all public solutions of $\Pi$.

**Algorithm 2:** Multiagent planning algorithm with complete PSMs

```
Function PsmPlanComplete($\Pi$) is
    foreach $\alpha \in \mathsf{agents}(\Pi)$ do
        $\Gamma_\alpha \leftarrow$ complete PSM of $\Pi \triangleright \alpha$;        // BFS or DFS search
        $\Delta_\alpha \leftarrow \Gamma_\alpha \triangleright \star$;
    end
    return $\bigcap_{\alpha \in \mathsf{agents}(\Pi)} \Delta_\alpha$;
end
```

**Theorem 2** *Let $\Pi$ be an* MA-STRIPS *problem and $\Delta =$* `PsmPlanComplete`$(\Pi)$. *It holds that* $\mathsf{accept}(\Delta) = \mathsf{sols}(\Pi) \triangleright \star$.

*Proof* Follows from Lemmas 2, 4, 5, and Theorem 1. □

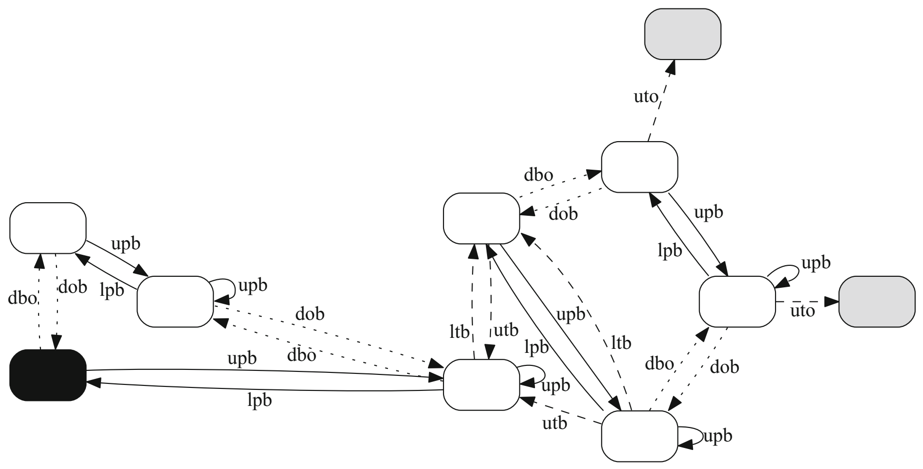

**Fig. 4** The complete PSM of agent *Truck* (Example 9)

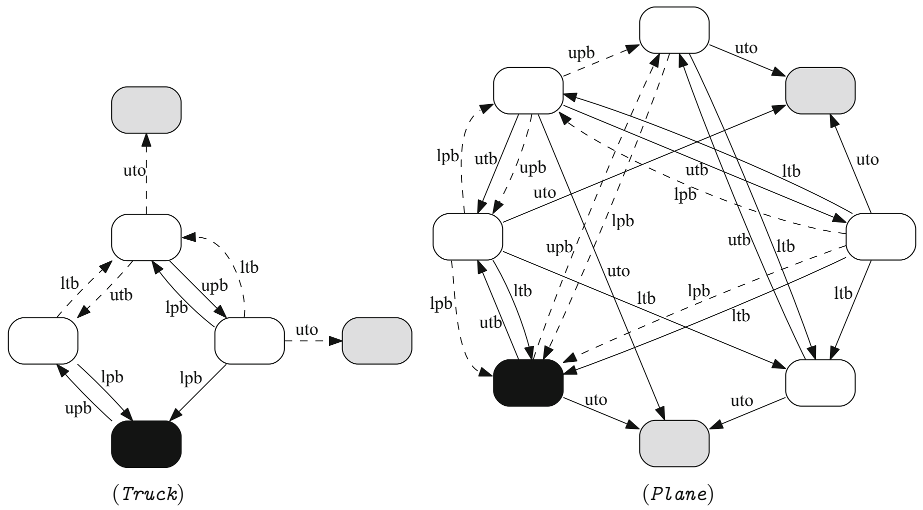

**Fig. 5** Public projections of complete PSMs of agents from Example 9

*Example 9* In this example, we demonstrate complete PSMs, public projection, and PSM's intersection on our running Example 1. The complete PSM of agent `Truck` is presented in Fig. 4. The *black node* represents the initial state, and the *gray nodes* are goal states. *Dotted edges* represent internal actions, *dashed edges* public actions, and *solid edges* external actions. To improve clarity, we shorten action labels by the first letters of involved objects, for example, the action `fly(prague, brno)` is shortened as "fpb", and so on. We also remove edges outgoing goal states, and we omit state labels which can be easily filled in using state progression function $\gamma$.

The complete PSM of agent `Plane` is too large for presentation (containing 32 states and 72 transitions). However, its public projection is shown together with the public projection of `Truck`'s public PSM in Fig. 5. Note how public projection decreases number of states and

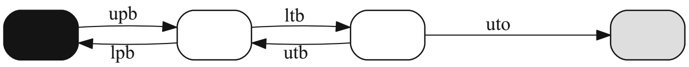

**Fig. 6** Intersection of public PSMs from Fig. 5 representing all possible public solutions

transitions. The intersection of both public PSMs is shown in Fig. 6. Note that the intersection PSM represents infinite set of all possible public solutions by a finite structure. □

## 5 Distributed PSM planner

This section describes how to use planning state machines to effectively solve multiagent MA-Strips problems. Section 4.5 already introduced planning algorithm which is correct and complete, and its advantage is that it computes all the public solutions of a given MA-Strips problem. However, its time and space complexity renders it unusable for more complex problems. In this section, we introduce an iterative algorithm based on the idea of PSMs which is designed to be usable in practice. The rest of this section describes the algorithm.

The basic idea can be described as follows. Every agent from the MA-Strips problem $\Pi$ executes its own planning loop perhaps on a different machine. Every agent $\alpha$ starts with an empty PSM $\Gamma_\alpha$. In every iteration, every agent generates a new plan solving its local problem $\Pi \triangleright \alpha$ and it adds this plan to $\Gamma_\alpha$. Then public projection $\Delta_\alpha = \Gamma_\alpha \triangleright \star$ is computed, and public PSMs are exchanged among the agents. This process continues until the intersection $\Delta = \bigcap_{\beta \in \mathsf{agents}(\Pi)} \Delta_\beta$ is not empty (meaning $\mathsf{accept}(\Delta) \neq \emptyset$). A non-empty intersection of public PSMs is thus guaranteed to contain at least one public solution of $\Pi$.

**Algorithm 3:** Distributed multiagent planning algorithm.

```
1  Function PsmPlanDistributed($\Pi \triangleright \alpha$) is
2      $\Gamma_\alpha \leftarrow$ empty PSM of problem $\Pi \triangleright \alpha$;
3      loop
4          generate new $\pi_\alpha \in \mathsf{sols}(\Pi \triangleright \alpha)$;                // Section 5.1
5          $\Gamma_\alpha \leftarrow \Gamma_\alpha \oplus \pi_\alpha$;
6          $\Delta_\alpha \leftarrow \Gamma_\alpha \triangleright \star$;
7          announce public PSM $\Delta_\alpha$ to other agents;
8          receive/update public PSMs of other agents;
9          $\Delta \leftarrow \bigcap_{\beta \in \mathsf{agents}(\Pi)} \Delta_\beta$;     // intersection of public PSMs of all agents
10         if $\mathsf{accept}(\Delta) \neq \emptyset$ then
11             return $\Delta$;
12         end
13         incorporate other agents $\Delta_\beta$ into local $\Pi \triangleright \alpha$;     // Section 5.1
14     end
15 end
```

The multiagent planning algorithm is described in Algorithm 3. By one *iteration* of the algorithm, we mean one execution of the loop (lines 3–14). By a new plan in the first step inside the loop, we mean a plan that was not generated in any of the previous loop iterations. To achieve this, we propose a technique based on known diverse planning techniques. Details are provided in Sect. 5.1. The variable $\Gamma_\alpha$ keeps a PSM representing all the plans generated so far. The new plan $\pi_\alpha$ is added to $\Gamma_\alpha$ and public PSM $\Delta_\alpha$ is computed using Algorithm 1. Then all public PSMs are exchanged among the agents. In our implementation, this is synchronization step when the executing agent might need to wait for other agents to finish their computations of public PSMs. However, an alternative non-blocking implementation where the executing agent only updates public PSMs currently available is also possible. Then the intersection $\Delta$ of public PSMs of all the agents is computed and possibly returned as a result. The intersection is computed by every agent to avoid a centralized component. In the last step of the loop, public PSMs of other agents are incorporated into the local planning problem $\Pi \triangleright \alpha$ using landmarks and action costs. This step, described in details in Sect. 5.2, is optional and can be skipped. Theorem 3 states the soundness and completeness of the algorithm.

**Theorem 3** (Completeness and soundness) *Let $\Pi$ be an* MA-STRIPS *problem such that* $\mathsf{sols}(\Pi) \neq \emptyset$. *Let $\Delta$ be a result of* `PsmPlanDistributed`$(\Pi \triangleright \alpha)$ *for an arbitrary agent* $\alpha$. *Then* $\mathsf{accept}(\Delta) \neq \emptyset$ *and* $\mathsf{accept}(\Delta) \subseteq \mathsf{sols}(\Pi) \triangleright \star$. *Moreover, if the underlying planner (1) is complete, (2) is optimal (with respect to length of generated plan), and (3) allows to generate different plans, then the algorithm always terminates.*

*Proof* Let $\pi$ be a solution of $\Pi$. As a result of completeness and optimality and of the underlying planner, agent $\alpha$ will generate, in the worst case, all the solutions of $\Pi \triangleright \alpha$ up to the length of $\pi$. These must include a solution with the public projection $\pi \triangleright \star$. As this hold for every agent $\alpha$, the intersection of their respective public PSMs must be non-empty and the algorithm terminates. This ensures algorithm completeness and termination. The soundness of the algorithm directly follows from the properties of intersection, Theorem 1, and Lemma 1. □

## 5.1 Generating new plans

This section describes how to generate a plan of a classical STRIPS problem which differs from a set of plans provided as an input. This extension is inspired by diverse planning with *homotopy class constraints* [3]. In our setting, homotopy classes of plans are naturally defined by plan public projections. That is, two plans $\pi_1$ and $\pi_2$ belong to same homotopy class iff $\pi_1 \triangleright \star = \pi_2 \triangleright \star$.

In our implementation, we have extended the FastDownward planner, but the same technique can be used to extend any planner based on a state-space search. The technique is based on the idea of augmented graphs [3]. Every state is extended by a vector of numbers where each vector field corresponds to one of the forbidden plans. The $i$th vector field value indicates whether the plan ending at this state is different from the $i$th forbidden plan. Value $-1$ indicates that current plan differs, while a nonnegative number denotes the position in the corresponding forbidden plan. During action application at some state, we check whether the applied action equals the expected action at the next position in the forbidden plan. The initial state corresponds to the vector of zeros. During the state search, a plan is accepted as a solution only when the plan ends in a state with only $-1$ values.

Algorithm 3 starts with an empty set of forbidden plans. In every iteration, the generated plan is added to this set. This ensures that the algorithm generates a different plan in every iteration.

As an optimization, we use action costs to force the underlying planner to prefer internal action to public actions, and public actions to external actions. These action costs[3] correspond

[3] We have chosen costs 10 for internal actions, 100 for public actions, and 1000 for external action. Nevertheless, the exact values are not important.

to the knowledge an agent has about these actions. An agent has full information about its internal actions, and as they do not affect other agents, they should be used whenever possible. The agent also has full information about its public actions, but because they can affect other agents they should be used more carefully. Finally, the agent has only a limited information about external actions of other agents because some information can be pruned of by public projection. Hence, external actions are the most expensive.

## 5.2 Guiding plan search using public PSMs

In every iteration of Algorithm 3, the agent receives public PSMs of all other agents. These PSMs contain information about plans found by other agents. Information in these PSMs can be used to guide a new plan generation so that the algorithm finds a solution faster. We incorporate information from public PSMs into the local planning problem by extending the problem with *soft-landmark* actions and by adjusting *action costs*. This problem extension influences plan search in the desired way. When agent $\alpha$ receives public PSM $\Delta_\beta$ of another agent $\beta$, we would like the local plan generator to prefer sequences of public actions suggested by $\Delta_\beta$. This is because Algorithm 3 terminates only when all the agents generate the same public solution. Hence, it is preferable to find a local solution $\pi_\alpha \in \mathsf{sols}(\Pi \triangleright \alpha)$ such that the public projection $\pi_\alpha \triangleright \star$ is contained in $\mathsf{accept}(\Delta_\beta)$. We achieve this by extending $\Pi \triangleright \alpha$ with special landmark actions without affecting the set of solutions of $\Pi \triangleright \alpha$. These landmark actions basically duplicate actions from $\Pi \triangleright \alpha$ but have decreased action costs. The landmark actions have additional preconditions to ensure that landmark actions are used in the order given by $\Delta_\beta$.

The process of extending local problem $\Pi \triangleright \alpha$ with $\Delta_\beta$ is sketched as follows. We extend the local problem with a set of fresh facts $P_{marks}$ distinct from facts of $\Pi \triangleright \alpha$ where each fresh fact corresponds to a state of $\Delta_\beta$. Hence, we have bijection $\mu$ from states of $\Delta_\beta$ to $P_{marks}$. Let $\Delta_\beta$ contain a transition from $s$ to $s'$ labeled by action $a$. We then extend the local problem with a duplicate of $a$ which (1) can be applied only in states where $\mu(s)$ is valid and (2) additionally transforms fact $\mu(s)$ to $\mu(s')$. In this way, landmark actions can be applied only in the order given by $\Delta_\beta$. The following defines a landmark action.

**Definition 13** Let $a$ be an action and let *from* and *to* be two facts. The *landmark action* $\mathsf{lm\text{-}act}(a, \mathit{from}, \mathit{to})$ is defined as follows.

$$\mathsf{lm\text{-}act}(a, \mathit{from}, \mathit{to}) = \langle \mathsf{id}(a), \mathsf{pre}(a) \cup \{\mathit{from}\}, \mathsf{add}(a) \cup \{\mathit{to}\}, \mathsf{del}(a) \cup \{\mathit{from}\} \rangle$$

For action id *id*, the landmark action (w.r.t. agent $\alpha$) is defined as

$$\mathsf{lm\text{-}act}_\alpha(id, \mathit{from}, \mathit{to}) = \mathsf{lm\text{-}act}(a, \mathit{from}, \mathit{to})$$

where $a$ is the uniquely determined action of $\Pi \triangleright \alpha$ with $\mathsf{id}(a) = id$. □

Once we have added landmark facts and actions, we just extend the initial state of the local problem with $\mu(I_0)$ where $I_0$ is the initial state of $\Delta_\beta$. A complete extension of the local problem is formally defined as follows.

**Definition 14** Let $\Pi$ be an MA-STRIPS problem and $\alpha \in \mathsf{agents}(\Pi)$. Let $\Delta = \langle \Sigma, S, I', \delta, F \rangle$ be a public PSM of $\Pi$. Let $P_{marks}$ be a set of facts distinct from $\mathsf{facts}(\Pi)$ such that $|P_{marks}| = |S|$ and let $\mu$ be a bijection from $S$ to $P_{marks}$. The problem $\Pi \otimes_\alpha \Delta$ is defined as the STRIPS problem $\langle P_0, A_0, I_0, G \rangle$ where

(1) $P_0 = \mathsf{facts}(\Pi \triangleright \alpha) \cup P_{marks}$, and
(2) $A_0 = \mathsf{actions}(\Pi \triangleright \alpha) \cup \{\mathsf{lm\text{-}act}_\alpha(id, \mu(s), \mu(s')) : s' \in \delta(s, id)\}$, and
(3) $I_0 = \mathsf{init}(\Pi \triangleright \alpha) \cup \{\mu(I')\}$.

The problem $\Pi \otimes_\alpha \Delta$ is called the *local problem of* $\alpha$ *extended with* $\Delta$. □

Note that the extension does not affect the goal state condition. There is a straightforward relationship between solutions of $\Pi \triangleright \alpha$ and solutions of $\Pi \otimes_\alpha \Delta_\beta$. Clearly every solution of $\Pi \triangleright \alpha$ is a solution of the extended problem $\Pi \otimes_\alpha \Delta_\beta$. On the other hand, every solution of $\Pi \otimes_\alpha \Delta_\beta$ can be translated to the solution of $\Pi \triangleright \alpha$ by replacing landmark actions with original actions of $\Pi \triangleright \alpha$.

The crucial point is that landmark actions are assigned significantly decreased action costs so that the underlying planner prefers landmark actions to original actions. In our implementation, the costs are set as follows. Suppose that $\Pi \triangleright \alpha$ is being extended with $\Delta_\beta$. The cost of $\mathsf{lm\text{-}act}(a, \mathit{from}, \mathit{to})$ is set to 1 if $a$ is owned by $\alpha$ (the receiver of $\Delta_\beta$) or if $a$ is owned by $\beta$ (the sender). Otherwise, the cost 10 is used.

The reasoning behind this choice is that the owner (either $\alpha$ or $\beta$) of the action has full information about it and thus it is more likely to be used correctly.

Definition 14 can be easily adjusted so that it allows repeated extension of $\Pi \triangleright \alpha$. During planning, problem $\Pi \triangleright \alpha$ is extended with every $\Delta_{\beta_i}$ received from other agents in the previous loop iteration, one by one. Hence, the underlaying planner is launched with the problem

$$\Pi \otimes_\alpha \Delta_{\beta_1} \otimes_\alpha \Delta_{\beta_2} \otimes_\alpha \cdots \otimes_\alpha \Delta_{\beta_m}$$

provided $\otimes_\alpha$ is left associative. In the first iteration, the local problem $\Pi \triangleright \alpha$ is used without any extension.

Practical experiments revealed that the actions costs and landmark actions described in this section are crucial for a practical usage of Algorithm 3. For experimental evaluation, see Sect. 8.

## 6 Improving PSM planner performance

In this section, we propose two methods to improve performance of multiagent planning Algorithm 3. The first method, which we call PSM-V, is based on plan verification, and it is described in Sect. 6.1. The second method, which we call PSM-R, computes a relaxed solution $\pi$ of a given MA-STRIPS problem, and its public projection $\pi \triangleright \star$ is used as an initial landmark by all the agents. Details are provided in Sect. 6.2. Both the methods can also be combined and used together. The combination of the methods is denoted PSM-RV. Section 8 evaluates the impact of the methods both when used separately and when used together.

### 6.1 Plan verification and analysis

Distributed PSM planner from Algorithm 3 uses public plans generated by other agents as landmarks to guide future plan search (see Sect. 5.2). However, it is desirable to use only extensible plans to guide plan search because non-extensible plans cannot lead to a non-empty public PSMs intersection. Every generated plan should be verified by other agents in order to determine its extensibility. However, extensibility (or $\alpha$-extensibility) checking is expensive, and thus, we propose only an approximative method of plan verification. An extension of a PSM planner with a method of plan verification has already been proposed with promising results [17]. The rest of this section describes the method.

First we describe how to approximate $\alpha$-extensibility of public plan $\sigma$. Given a public plan $\sigma = \langle id_1, \ldots, id_n \rangle$, we create a problem $\Pi \circledast_\alpha \sigma$ which is solvable iff $\sigma$ is $\alpha$-extensible. We extend the set of facts with fresh facts $P_{marks} = \{m_0, \ldots, m_n\}$. We add the action $\mathsf{lm\text{-}act}(a_i, m_{i-1}, m_i)$ for all $0 < i \leq n$ to the actions of $\Pi \circledast_\alpha \sigma$ (see Definition 13). The initial state of $\Pi \circledast_\alpha \sigma$ is extended with $m_0$, but, this time, the last mark fact $m_n$ is added to the goal of $\Pi \circledast_\alpha \sigma$. This ensures that any solution of $\Pi \circledast_\alpha \sigma$ contains all actions from $\sigma$ in the right order, possibly interleaved with $\alpha$'s internal actions. Hence, every solution of $\Pi \circledast_\alpha \sigma$ can be translated to a solution of $\Pi \triangleright \alpha$ (but not necessarily the other way round). The following formalizes construction of $\Pi \circledast_\alpha \sigma$ and proves its correctness.

**Definition 15** Let $\Pi$ be a MA-STRIPS problem and $\alpha \in \mathsf{agents}(\Pi)$. Let $\sigma = \langle id_1, \ldots, id_n \rangle$ be a public plan of $\Pi$. Let $P_{marks} = \{m_0, \ldots, m_n\}$ be a set of facts distinct from $\mathsf{facts}(\Pi)$. The *$\alpha$-extensibility check problem of* $\sigma$, denoted $\Pi \circledast_\alpha \sigma$, is the STRIPS problem $\langle P_{marks} \cup (\mathsf{facts}(\Pi) \triangleright \alpha), A, (\mathsf{init}(\Pi) \triangleright \alpha) \cup \{m_0\}, \mathsf{goal}(\Pi) \cup \{m_n\} \rangle$ where $A = \mathsf{int\text{-}actions}(\alpha) \cup \{\mathsf{lm\text{-}act}_\alpha(id, m_{i-1}, m_i) : 0 < i \leq n\}$. □

**Theorem 4** ([35]) *Let $\Pi$ be an* MA-STRIPS *problem, let $\alpha$ be an agent of $\Pi$, and let $\sigma$ be a public plan of $\Pi$. Then $\sigma$ is $\alpha$-extensible iff* $\mathsf{sols}(\Pi \circledast_\alpha \sigma) \neq \emptyset$.

*Proof* Both the implications are proved similarly by replacing public actions with their respective landmark actions ($\Rightarrow$) or the other way round ($\Leftarrow$). □

A previous attempt to use the above construction of $\Pi \circledast_\alpha \sigma$ can be found [35]. It tries to centrally generate public solutions $\sigma$ and to verify that $\sigma$ is $\alpha$-extensible by every agent $\alpha$. A roadblock of this attempt is that it is relatively hard for agent $\alpha$ to find out that $\sigma$ is *not* $\alpha$-extensible. Then $\mathsf{sols}(\Pi \circledast_\alpha \sigma) = \emptyset$, and it usually requires the underlaying planner used to solve $\Pi \circledast_\alpha \sigma$ to traverse the whole search space. That is because the state-of-the-art planners are optimized to find a solution of a given problem and not to determine that the problem is not solvable. That is why we have proposed [17] an approximate method to determine problem solvability.

Previously proposed approximation of $\Pi \circledast_\alpha \sigma$ solvability [17] is done using generic process calculi type system scheme POLY✶ [18,25]. The same result can be achieved using *planning graphs* [14, Chapter 6] which we briefly describe here. We construct a complete relaxed planning graph of $\Pi \circledast_\alpha \sigma$, that is, the planning graph of $\Pi \circledast_\alpha \sigma$ with action delete effects removed. A planning graph with $k$ layers can be constructed in time polynomial in $k$. Then we examine the last fact layer of the constructed planning graph. Recall that $\Pi \circledast_\alpha \sigma$ contains fresh mark facts $P_{marks} = \{m_0, \ldots, m_n\}$. When mark $m_i$ is valid in some planning state, it means that (1) public actions $a_1, \ldots, a_i$ from $\sigma$ were already correctly used in the current plan and that (2) the next public action to be used is $a_{i+1}$. The result of the analysis is the maximum $j$ such that $m_j$ is in the last fact layer of the relaxed planning graph. This resulting $j$ is interpreted as follows. When $j < n$, clearly $\Pi \circledast_\alpha \sigma$ is unsolvable because $m_n$ is a goal fact. Moreover, the result $j$ tells us that there is no way for an agent to follow the public plan $\sigma$ up to the point where $a_{j+1}$ can be applied. This gives us an approximation of a valid prefix of $\sigma$. On the other hand, $j = n$ does not necessarily implies that $\sigma$ is $\alpha$-extensible because the proposed method is only an approximation of $\Pi \circledast_\alpha \sigma$ solvability.

The above $\alpha$-extensibility approximation allows the following plan verification procedure. When agent $\alpha$ generates a new plan $\pi_\alpha$, it sends its public projection $\pi_\alpha \triangleright \star$ to all the other agents. Once other agent $\beta$ receives $\pi_\alpha \triangleright \star$, it runs the above $\beta$-extensibility check and sends its result back to agent $\alpha$ (just after line 4 of Algorithm 3). Agent $\alpha$ collects analysis results from all the other agents and computes their minimum $l$. Plan $\pi_\alpha$ is then stripped so that only the first $l$ public actions remain in it. This stripped plan is then used to extend PSM $\Gamma_\alpha$ in Algorithm 3. Hence, only the stripped plan is used as a landmark to guide future plan search. In the next iteration, a plan with public projection different from $\pi_\alpha$ is required to be computed. When $\pi_\alpha \triangleright \star = \langle id_1, \ldots, id_n \rangle$, we can even further speed up convergence of Algorithm 3 by forbidding any plan with public prefix $\langle id_1, \ldots, id_l, id_{l+1} \rangle$ to be generated in the future. This does not affect completeness of Algorithm 3 (Theorem 3) because only provably non-extensible plans are forbidden.

*Example 10* In this example, we demonstrate planning with plan analysis on our running Example 1. The *first iteration* is as follows. All the agents use an optimal planner in order to solve their local problems and thus agent *Plane* creates the simplest plan where the goal is reached by agent *Truck* alone. That is, *Plane* generates the plan $\langle$`unload(truck, ostrava)`$\rangle$ (see also Example 5). This plan does not pass through the verification process of agent *Truck* because *Truck* needs to execute additional public actions prior to the last goal-reaching action. In the meanwhile, agent *Truck* generates a plan with following public projection.

$\sigma = \langle$`unload(plane, brno), load(truck, brno), unload(truck, ostrava)`$\rangle$

This public plan is extensible and thus passes through the verification check. Landmark actions created from $\sigma$ are added to *Plane*'s local problem. The intersection of public PSMs after the first iteration is empty because *Plane*'s PSM is empty.

In the *second iteration*, *Plane* follows the landmarks from the above public plan $\sigma$ and it succeeds by generating a plan with the public projection $\sigma$. At this point, public PSMs of both the agents contain $\sigma$ and hence the algorithm terminates after the second iteration independently on the plan generated by *Truck*. □

## 6.2 Initial relaxed plan landmark

The delete effect relaxation, where delete effects of actions are ignored, has proved its relevance both in STRIPS planning [15] and recently also in MA-STRIPS planning [32]. It is known that to find a solution of a relaxed problem is an easier task than to find a solution of the original problem. There are distributed algorithms to find a solution of a relaxed MA-STRIPS problem using *distributed planning graphs* [31]. Effective implementation using *exploration queues* can also be found in the literature [32]. All these algorithms respect privacy, that is, they do not reveal internal facts and actions to other agents.

We use a relaxed solution of MA-STRIPS problem $\Pi$ to improve Algorithm 3 as follows. At first we compute some solution $\pi$ of the relaxation of $\Pi$. We compute its public projection $\pi \triangleright \star$ which is a sequence of public action ids. We use this id sequence as an initial landmark in the first loop iteration of Algorithm 3. The sequence is integrated into $\Pi \triangleright \alpha$ in the same way as in Definition 14 (the public projection $\pi \triangleright \star$ can be seen as a public PSM which contains only $\pi \triangleright \star$). The same initial landmark is used by all the agents. When $\pi \triangleright \star$ is extensible, every agent $\alpha$ is likely to generate local solution $\pi_\alpha$ such that $\pi_\alpha \triangleright \star = \pi \triangleright \star$ in the first iteration. In that case, the algorithm terminates directly in the first iteration causing a dramatic speedup. Otherwise, the initial landmark is forgotten by all the agents and the algorithm continues by the second iteration as before. Practical impact of initial relaxed plan landmarks is experimentally evaluated in Sect. 8.

*Example 11* In our running Example 1, we can find a relaxed solution and compute its public projection. The shortest solution has the following public projection.

$\sigma = \langle$`unload(plane, brno), load(truck, brno), unload(truck, ostrava)`$\rangle$

We can see that this public plan $\sigma$ is extensible. When the agents uses $\sigma$ as landmarks in the first iterations, both of them succeed to generate a plan with public projection $\sigma$. Hence, the intersection of public PSMs is not empty and the algorithm terminates after the first iteration. $\square$

## 7 From theory to implementation

In this section, we describe several interesting problems encountered when implementing a PSM-based planner. Firstly, the theory is based on MA-STRIPS formalism which is based on STRIPS. Just like STRIPS, MA-STRIPS requires *grounded* problem specification which is not appropriate for real-world problems. Section 7.1 describes how to move toward more convenient PDDL-like planning language which allows a compact representation by introduction of *parametric* actions. Then, in Sect. 7.2, we look on formalisms extending single-agent PDDL to multiagent planning problems focusing mainly on definition of privacy of agent knowledge. Finally, in Sect. 7.3, we describe how to handle internal goals without the need to publish them.

### 7.1 From STRIPS to PDDL, and back again

STRIPS language is a formal language often used in automated planning theory. STRIPS also provides a formal base for MA-STRIPS. Nevertheless, it is not very practical for real-world problems because it supports only grounded representation. Therefore, in practice and in benchmark tests, planning domain definition language (PDDL) is typically used. PDDL supports predicates with typed parameters which allow to describe a full range of facts or actions by parametric statements. When we convert our running Example 1 into PDDL language, we can have only one parametric action `drive`(*from*, *to*) instead of multiple actions for different locations. But backward grounding of this parametric action can create instances which were not in the original domain. To get rid of these instances, new predicate `isRoad`(*from*, *to*) can be introduced. Such a predicate is never part of action effects, and thus, it is *constant* during execution of any plan. When we ground a PDDL problem, we can evaluate these predicates and we can omit action instances where the predicate evaluates to *false* because these instances can never be used. Using the same definition of public facts for PDDL as we described in MA-STRIPS, these constants would be public, because they appear as precondition of actions of different agents. Nevertheless, it is not necessary to communicate them because their evaluation never changes and thus every agent can has its own copy.

A conversion from PDDL to STRIPS, that is, *grounding*, is needed in two places. Firstly, we use it to compute the size of a problem because number of predicates and actions of the grounded problem better describes real complexity of the problem. More importantly, it is needed to compute MA-STRIPS representation of input problems because FMAP problems, which we use as benchmarks (see Sect. 8), are defined in the PDDL format. Hence, input PDDL problem needs to be grounded so that we can use MA-STRIPS definition of fact privacy classification. FMAP benchmark problems define a separate domain and problem PDDL file for every agent. Firstly, we create a single-agent problem by merging all agent domains and problems. Secondly, we use grounding algorithm implemented in FastDownward planner[4] to ground it. Then we take all the facts used in the grounded problem and ground the original agents' problems to these facts.

[4] See http://www.fast-downward.org/. Script `translate.py` creates an SAS representation of an input PDDL problem.

### 7.2 Multiagent PDDL extensions

STRIPS and PDDL are two standard languages to describe deterministic single-agent planning problems. Nevertheless, there is no similar standard in the multiagent planning. There are several attempts to create such a standard: NADL [19], concurrent interacting actions in STRIPS [5], MAPL [8], FMAP [34], MA-PDDL [23], concurrent STRIPS [29], and MA-STRIPS [6].

In our work, we focus on PDDL which allows to describe agent actions and a state of the world more naturally. PDDL is also used as a standard in international planning competition whose planning problems are used as standard banchmarks in planning community. PDDL extensions which we have considered in our work are as follows.

FMAP—each agent has its own domain and problem file. PDDL language is extended with $\texttt{shared} - \texttt{data}$ field which allows to specify which predicates are shared with which agents. However, this approach does not always allow to define fact privacy classification as defined by MA-STRIPS.

MA-PDDL—extension of PDDL 3.1 which allows to specify the owner of each action using field `agent`. Nevertheless, it does not allow to manually specify what facts are public/internal.

None of the existing PDDL extensions allow to express both FMAP (public facts specified by a list of public predicate names) or MA-STRIPS (see Sect. 3) privacy definitions. Hence, we propose our extension which allows us to finely tune the amount of knowledge shared among the agents up to the level of single facts.

### 7.3 Problems with internal goals

So far we have considered MA-STRIPS problems where all goal facts are public. This can always be achieved by publishing of internal goal facts. Nevertheless, in some cases, agents can have different internal goals and agents might not be willing to share these facts with other agents out of privacy concerns. We still consider cooperating agents. This section describes how to transform an MA-STRIPS problem with internal goals to an equivalent problem where the original internal goals can be kept internal.

The transformation extends each agent $\alpha$ with a public *confirmation action* which can be executed when all the goals of $\alpha$ are satisfied. The confirmation actions can be executed only at the end of a plan. The goal of the transformed problem expresses that all the agents confirmed their goals.

More formally, let $\Pi$ be an MA-STRIPS problem where some or all goals are internal. Firstly, we introduce a fresh fact `planning` with the meaning that planning is in progress, that is, that no confirmation action (of any agent) has been used so far. Fact `planning` shall be initially valid in the initial state of the transformed problem and shall be deleted by the first confirmation action. Every action from $\Pi$ shall be extended with precondition `planning`. Next we introduce fresh virtual goal facts $\texttt{done}_1, \ldots, \texttt{done}_n$. Fact $\texttt{done}_i$ means that the confirmation action of the $i$th agent was used. The confirmation action $\mathsf{confirm}(\alpha, i)$ of agent the $i$th agent $\alpha$ can be used when all the goals relevant for $\alpha$ (i.e., public goals and $\alpha$'s internal goals) are satisfied. Its add effect is the virtual goal $\texttt{done}_i$, and it deletes `planning`, formally, $\mathsf{confirm}(\alpha, i) = \langle \mathsf{goal}(\Pi) \triangleright \alpha, \{\texttt{done}_i\}, \{\texttt{planning}\}\rangle$. The goal of the transformed MA-STRIPS problem is set to $\{\texttt{done}_1, \ldots, \texttt{done}_n\}$.

There is a direct correspondence between solutions of the original and of the transformed problem. Every solution of the transformed problem can be translated to a solution of the original problem by removing confirmation actions. Moreover, the goal facts of the transformed problem can be freely published because they do not carry any confidential information.

## 8 Experimental results

We have performed a set of experiments to compare our planners with another state-of-the-art multiagent planners[5] and also to evaluate the impact of plan verification on planning times. We have decided to compare our planners with FMAP [34], RDFF [32], and GPPP [26]. All planners are compared on well-defined problems taken from international planning competition (IPC) problems as published by FMAP authors. FMAP classifies facts as public or internal using a manual selection of public predicate names. On the other hand, RDFF and GPPP use privacy classification as defined by MA-STRIPS. In practice, FMAP public facts are a superset of MA-STRIPS public facts, and our PSM-based algorithms can handle both privacy classification. In our experiments, we use exactly the same input files as the authors of FMAP used during its evaluation,[6] and we also use the same time limit of 30 min for each problem.

The above-mentioned state-of-the-art multiagent planners are compared with several variants of our PSM-based planner. Variant PSM is the basic version described by Algorithm 3 in Sect. 5. Then we have two extensions of this basic algorithm. Variant PSM-R uses initial relaxed plan landmarks described in Sect. 6.2. Variant PSM-V is the basic algorithm extended with the plan verification as described in Sect. 6.1. Finally, there is PSM-RV which combines both extensions into a solid planner that benefits from of all these features.

The experiments are organized as follows. Firstly, we describe benchmark domains in Sect. 8.1. Then, in Sect. 8.2, we compare the variants of our algorithm and compare it with FMAP planner which has currently the highest coverage between multiagent planners. In Sect. 8.3, we further analyze the communication of the PSM variants and explore its connection with the number of iteration needed to solve a problem. The experiments are concluded in next Sect. 9 where we focus on different privacy classifications. We compare our algorithm with MA-STRIPS-based planners and show how the increase of privacy of facts and goals affects performance of our planner.

### 8.1 Benchmark domains

We have performed experiments on 244 PDDL problems from 10 domains described in Fig. 7. These problems, published by the authors of FMAP, are inspired by traditional benchmark problems of international planning competition (IPC).[7] The privacy classification is defined as a list of predicates shared with other agent. It is therefore possible to specify that some knowledge is shared between two agents only. However, in these problems all knowledge is always shared among all the agents. Goals are always public. Let us call this set of problems FMAP problems and refer to this level of privacy as FMAP privacy.

In FMAP problems, constants known to all agents are often considered to be internal. For example, every agent knows that there is a road between two cities, but the agents do not know that other agents know it. Thus, for example in *Driverlog* domain, every *truck*

[5] All the tests were performed on a single PC, CPU Intel i7 3.40 GHz with 8 cores, and memory limited to 8 GB RAM.

[6] We would like to thank the authors of FMAP for a kind support with their planner.

[7] http://ipc.icaps-conference.org/.

*Blocksworld* is a multiagent version of the classical planning problem where each of 4 agents represents one robotic arm that can move and stack blocks. All information in this domain is public.
*Depots* problems contain two types of agents. *Trucks* transport crates between hoists located in *depots*. These *depots* move crates to correct pallets. *Depots* problems contain from 5 to 12 agents. All information is public.
*Driverlog* problems contain from 2 to 8 agents representing *drivers* that operate several *trucks* to transport packages to required locations. All information is known by all agents. Nevertheless, some constants are internal (`link` and `path` representing roads and paths connecting different locations).
*Elevators* contain two types of agents representing *slow* and *fast elevators*. The goal is to transport passengers between floors. *Elevators* problems contain from 3 to 5 agents. Positions of passengers are always public, while elevator positions and the number of passengers in each elevator are internal. Passenger positions are changed by actions `board` and `leave` with natural meaning.
*Logistics* domain contains two types of agents, *trucks* and *planes*, transporting packages between cities. Loading and unloading of packages is performed by actions `load` and `unload`, respectively. A goal specifies only the final location of packages. A transportation task often requires cooperation of several agents. *Logistics* problems contain from 3 to 10 agents. The location of an agent is internal but the location of a package is public.
*Openstacks* problems contain a *manager* agent who handles product orders, and *manufacturer* agents who produce these products. This problem is based on minimum maximum simultaneous open stacks combinatorial optimization problem. All information is public including goals specifying that the orders have been shipped.
*Rovers* problems contain from 1 to 8 *rovers*, each represented by one agent. The goal is to collect samples and communicate acquired data. Every *rover* is capable of fulfilling of an arbitrary goal but an agent has limited resources and thus it is necessary to decide which goal will be fulfilled by which agent. Sample locations and information about whether the data have been communicated is public.
*Satellites* problems contain from 1 to 12 *satellite* agents taking images in space. The pointing of a satellite and whether an image has been taken is public. Both can be included in a goal. Predicate `pointing` is also occasionally part of the goal.
*Woodworking* domain contains 4 agents representing 7 machines in a production chain. All information is public.
*Zenotravel* domain contains from 2 to 8 agents representing *planes* with a limited fuel. The goal is to transport passengers between cities but it can also specify positions of some planes. Positions of passengers and planes are public. A fuel level is internal. Thus, all `fly` actions are public and only `refuel` actions are internal.

**Fig. 7** FMAP benchmark domains description

publishes its `drive` action without precondition that there is a road connecting two cities. In the descriptions bellow, when it is stated that "*All information is public*," the problem can contain these internal constants.

## 8.2 Overall benchmark results

Table 1 shows an overall coverage of solved problems. We can see that the FMAP has better results in most of the domains and also in the overall coverage when compared with basic PSM, PSM-R and PSM-V variants. Nevertheless, we can see that both PSM-R and PSM-V excel in few domains (PSM-R in *Elevators* and *Logistics*, and PSM-V in *Rovers*). PSM extended with both features—PSM-RV—keeps the benefits of these features and outperforms FMAP in the overall score.

A relaxed plan helps especially in *Elevators* and *Logistics* domains. In both domains, the relaxed plan well captures the coordination points, that is, where the *passenger* (or *package*) will be transported by which agent (*elevator* or *truck*). The relaxed plan thus represents a solution outline which is then extended by each agent with internal actions. This allows to solve the task in a single iteration.

**Table 1** Number of problems solved by the compared planners

| **Domain** | FMAP | PSM | PSM-R | PSM-V | PSM-RV |
|---|---|---|---|---|---|
| **Blocksworld** (34) | 19 | **27** | 26 | 26 | 26 |
| **Depots** (20) | **6** | 0 | 0 | 0 | 3 |
| **Driverlog** (20) | **15** | 10 | 13 | 14 | 14 |
| **Elevators** (30) | **30** | 1 | **30** | 4 | **30** |
| **Logistics** (20) | 10 | 0 | **20** | 0 | **20** |
| **Openstacks** (30) | 23 | **30** | **30** | **30** | **30** |
| **Rovers** (20) | **19** | 7 | 6 | 14 | 16 |
| **Satellite** (20) | **16** | 6 | 6 | 9 | 9 |
| **Woodworking** (30) | 22 | **27** | **27** | **27** | **27** |
| **Zenotravel** (20) | **18** | 17 | **18** | 17 | **18** |
| **Total** (244) | 178 | 125 | 176 | 141 | **193** |

Privacy classification follows FMAP and thus the results are not directly comparable with MA-STRIPS planners

Plan verification, represented by PSM-V variant, helped in *Driverlog* and *Rovers* problems where many solutions generated by an agent were unacceptable by some other agent. In *Driverlog*, this is caused by the privacy of constants defining the topology of the world (the predicates `link` and `path` describing connected locations). The convergence is improved by trimming out parts of plans which are impossible to fulfill, that is, a `drive` action between locations which are not connected by a road. In *Rovers*, the situation is similar—private constants describe abilities of each rover. When an agent requires some action from another agent which cannot be performed, the verification allows to remove such a plan from landmarks so that other agents are not confused by it.

Table 2 compares run times needed to solve selected tasks solvable by all the PSM variants. We can see that all the PSM variants scale much better than FMAP, especially in the *Openstacks* problems which are all solved in the first iteration. PSM performs best in most domains. This is a result of the requirement that the selected problems have to be solvable by all the variants and the basic PSM does not need to spend time on relaxed plan creation or on verification.

Left graph of Fig. 8 shows how much time it is needed to solve different problems by PSM-RV as a function of problem size. The problem size is calculated as a number of actions and facts of the grounded problem (described in Sect. 7.1). Right graph of Fig. 8 shows the time spent during the verification of other agents' plans. It shows that an agent spends less than one-third of its computation time on verification. In average, it is approximately 14 % of agent computation time. The relative time needed for verification is independent on the problem size, and its grow/descent depends on a particular domain.

## 8.3 Communication overhead evaluation

Figure 9 shows an average number of iterations required to solve problems of selected domains (left), and the amount of communication among the agents (right). The averages are taken only for the problems solved by all three variants: PSM-R, PSM-V, and PSM-RV. The numbers in parenthesis show how many problems is in this intersection (note that no *Depot* nor *Logistic* problem is solvable by PSM-R and thus the domains are not included). The performance measured by the number of iterations does not differ much over the domains (with the exception of *elevators* where the number of iterations of one problem is significantly decreased). The amount of communication is measured as the total number of actions sent among all agents. This includes the exchange of public plan projections used as landmarks and also queries for plan verification. As expected, we can see that verification requires addi-

**Table 2** Comparison of run times on selected problems solved by all the planners

| | FMAP | PSM | PSM-V | PSM-R | PSM-RV |
|---|---|---|---|---|---|
| **Driverlog** | | | | | |
| **p-01** | **0.6** | 2.2 (2) | 2.3 (2) | 1.1 (1) | 1.2 (1) |
| **p-05** | **1.8** | 33.8 (9) | 6.5 (3) | 3.2 (2) | 4.1 (2) |
| **p-08** | 11.9 | 9.6 (3) | 6.0 (2) | **4.2** (2) | 5.4 (2) |
| **p-10** | **2.1** | 3.0 (2) | 4.3 (2) | 3.5 (2) | 4.8 (2) |
| **p-13** | 16.2 | 13.7 (3) | 14.6 (3) | **8.2** (2) | 8.3 (2) |
| **Openstacks** | | | | | |
| **p-01** | 1.4 | 1.7 (1) | **1.2** (1) | **1.2** (1) | 1.4 (1) |
| **p-06** | 9.7 | 1.8 (1) | **1.7** (1) | 1.8 (1) | 3.1 (1) |
| **p-11** | 51.0 | **1.8** (1) | 2.3 (1) | 3.3 (1) | 5.7 (1) |
| **p-16** | 171.0 | **2.2** (1) | 4.4 (1) | 5.5 (1) | 9.1 (1) |
| **p-21** | 497.0 | **2.4** (1) | 6.4 (1) | 8.7 (1) | 14.1 (1) |
| **p-26** | N/A | **2.9** (1) | 12.5 (1) | 13.3 (1) | 22.6 (1) |
| **Woodworking** | | | | | |
| **p-01** | 2.7 | **1.2** (1) | 2.0 (1) | **1.2** (1) | 2.1 (1) |
| **p-06** | 200.3 | 4.0 (2) | 10.2 (2) | **3.5** (2) | 7.4 (2) |
| **p-11** | 1.9 | **1.2** (1) | 1.4 (1) | **1.2** (1) | 1.5 (1) |
| **p-16** | N/A | **3.2** (2) | 5.7 (2) | 3.3 (2) | 5.7 (2) |
| **p-21** | **0.4** | 1.2 (1) | 1.3 (1) | 1.2 (1) | 1.5 (1) |
| **p-26** | N/A | **1.4** (1) | 2.1 (1) | 1.5 (1) | 2.7 (1) |

Times are in seconds; PSM variants have number of iterations in parenthesis

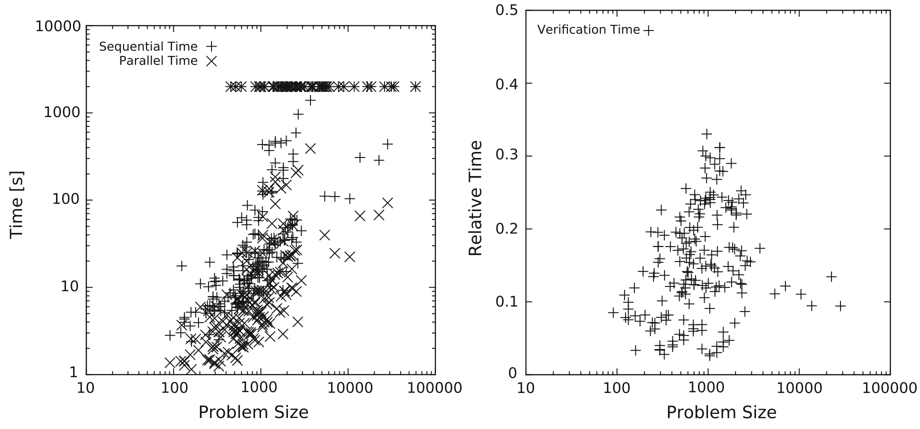

**Fig. 8** *Left* time needed to solve task as a function of problem size (log axis; time 2000 means not solved). *Right* portion of time an agent spends on verification of other agents' plans

tional communication, but in several domains this increase is outweighed by the decrease in number of iterations needed to solve the task.

## 9 Privacy analysis of planning problems

In this section, we analyze different privacy classifications (Sect. 9.1) and we compare benchmark domains with respect to the privacy classifications (Sect. 9.2). We experimentally

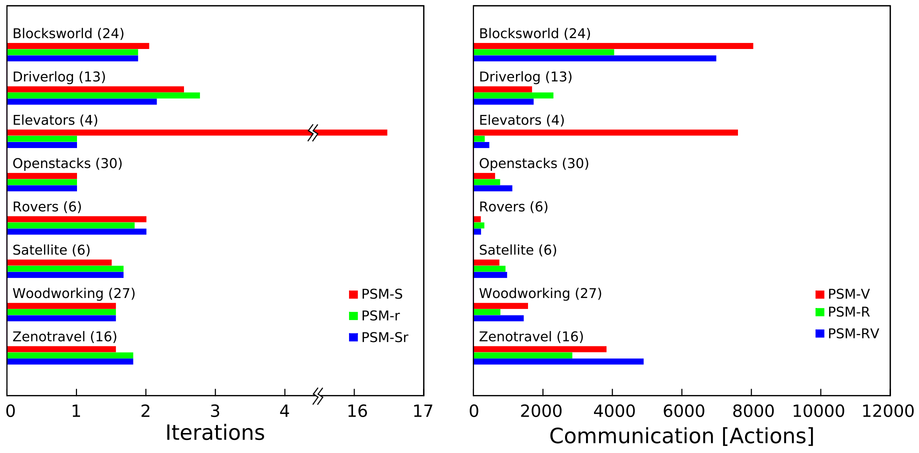

**Fig. 9** Number of iterations and amount of communication of different methods—always taken as average over problems solved by all methods in the graph (number of problems is appended to the domain name)

evaluate the impact of privacy classifications on the performance of our best PSM-based planner PSM-RV. We compare PSM-RV with other state-of-the-art MA-STRIPS planners (Sect. 9.3).

## 9.1 Privacy classifications

In Sect. 3 above, we have described MA-STRIPS privacy classification which defines the minimal amount of public information needed to be shared by different agents. MA-STRIPS requires facts used by actions of at least two different agents to be public. We have converted FMAP problems to MA-STRIPS problems (as described in Sect. 7.1) which reduces the amount of public information. Moreover, we allow agents to have internal goals which further increases privacy.

We now have four different privacy classifications for our benchmark problem set. First is the original FMAP privacy setting. Second, denoted FMAP$^{\ominus}$, is a variant of FMAP where those facts mentioned only by a single agent are made internal (which can make some goals internal). Third, denoted MA-STRIPS, is the MA-STRIPS privacy classification with public goals. Fourth, denoted MA-STRIPS$^{\ominus}$, is the MA-STRIPS classification which allows internal goals.

Let FMAP($\Pi$) denote the percentage of private facts in problem $\Pi$ with respect to FMAP classification, and similarly for the other three privacy classifications. It certainly holds that FMAP($\Pi$) $\leq$ FMAP$^{\ominus}$($\Pi$) and MA-STRIPS($\Pi$) $\leq$ MA-STRIPS$^{\ominus}$($\Pi$), and finally FMAP($\Pi$) $\leq$ MA-STRIPS($\Pi$).

## 9.2 Privacy in benchmark domains

Privacy of agent knowledge in different domains is in details described in Fig. 10.

Privacy classifications on benchmark domains, measured as a relative ratio of internal facts and actions, are demonstrated in Table 3.

## 9.3 Privacy benchmarks

In Sect. 8 above, we have compared our PSM-based planners with FMAP on problems with FMAP privacy classification. Now we compare our best PSM-RV with MA-STRIPS-based algo-

*Blocksworld* (Fmap) All information is public. (Fmap$^{\ominus}$) Information what an agent is holding is made internal (predicate `holding`). This predicate is never part of the goal and thus all goals are public. (MA-Strips) Facts that do not need to be shared are internal including all instances of predicate `holding`. Nevertheless, every action changes the position of a block which affects every other agent, and thus all the action are public. (MA-Strips$^{\ominus}$) Same as MA-Strips.

*Depots* (Fmap) All information is public. (Fmap$^{\ominus}$) The location of a *truck* is internal (predicate `at`). This predicate is never part of any goal and thus all goals are public. (MA-Strips) *Truck* locations and *truck* loads are always internal. The position of a crate is specified by predicate `on`. A crate can be either `on` a hoist or `on` a truck. When it is `on` a truck, the predicate is internal. (MA-Strips$^{\ominus}$) Same as MA-Strips.

*Driverlog* (Fmap) All information is known by each agent. Nevertheless, some constants are defined as internal (`link` and `path`). (Fmap$^{\ominus}$) *Driver* locations are internal (predicate `at`). Those parts of a goal specifying driver locations are internal. (MA-Strips) The location of a *driver* (predicate `at`) is internal if it is not part of a goal. Actions `walk` changing *driver*'s position are also internal unless they mention some public position. (MA-Strips$^{\ominus}$) Predicate `at` is always internal and the respective parts of goals are internal to some agent.

*Elevators* (Fmap) Positions of passengers are always public while *elevator* positions and the number of passengers in each *elevator* are internal. (Fmap$^{\ominus}$) Same as Fmap. (MA-Strips) Only passenger positions at floors accessible by two or more *elevators* and goal positions are public. Actions `board` and `leave` at these floors are public. All other information is internal. (MA-Strips$^{\ominus}$) Positions of passengers at floors accessible by only one *elevator* and respective `board`/`leave` actions are internal.

*Logistics* (Fmap) Agent locations are internal but the package location is public. (Fmap$^{\ominus}$) Same as Fmap. (MA-Strips) Locations of packages accessible to only one *truck* or *plane* are internal to appropriate agent. Actions `load` and `unload` at these locations are internal too. (MA-Strips$^{\ominus}$) Same as MA-Strips.

*Openstacks* (Fmap) All information is public including goals that the orders have been shipped. (Fmap$^{\ominus}$) Information about orders (predicates `waiting` and `shipped`), including all goals, is known to *manager* only. *Manufacturers* have empty goals. (MA-Strips) Instances of predicate `waiting` are internal but goal facts `shipped` are public. (MA-Strips$^{\ominus}$) Same as Fmap$^{\ominus}$.

*Rovers* (Fmap) Locations of samples and whether the data have been communicated are public. (Fmap$^{\ominus}$) Same as FMAP. (MA-Strips) If a sample can be analyzed by only one *rover* then the location of this sample is agent's internal fact. (MA-Strips$^{\ominus}$) Same as MA-Strips.

*Satellites* (Fmap) Pointings of satellites and whether the image has been taken is public – both is used by goals. (Fmap$^{\ominus}$) Pointings of satellites are internal (predicate `pointing`). This is occasionally internal part of a goal. (MA-Strips) Pointings of satellites are internal unless they appear in a goal. (MA-Strips$^{\ominus}$) Same as Fmap$^{\ominus}$.

*Woodworking* (Fmap) All information is public. (Fmap$^{\ominus}$) Constants and the internal state, specifying whether some board is loaded in the *highspeed saw* (predicate `in-highspeed-saw`), are internal. (MA-Strips) As in Fmap$^{\ominus}$. Moreover, *saw* agent always starts the production chain and thus the availability of resources is its internal knowledge (that is, predicate `available`). *Saw* agent has to perform at least two actions before it produces an intermediate product needed by other agents. Thus these initial actions are also internal (for example, action `load-highspeed-saw`). (MA-Strips$^{\ominus}$) Same as MA-Strips.

*Zenotravel* (Fmap) Positions of passengers and planes are public. A fuel level is internal. Thus, all `fly` actions are public and only `refuel` actions are internal. (Fmap$^{\ominus}$) Positions of planes are internal (predicate `at`). Thus, only positions of passengers are public. Therefore, actions `fly` and `zoom` (which is actually a shortcut for two `fly` actions) are internal. (MA-Strips) Same as Fmap$^{\ominus}$, except those goal facts that are public and the positions of passengers (predicate `in`) in cities reachable by only one plane, which are internal. (MA-Strips$^{\ominus}$) Similarly to Fmap$^{\ominus}$, positions of planes are always internal. Moreover, positions of passengers (predicate `in`) in cities reachable by only one plane are also internal.

**Fig. 10** Description of domains with extended privacy

**Table 3** Percentage of internal facts (left) and actions (right) in benchmark domains with respect to different privacy classifications

| | FMAP (%) | | FMAP$^{\ominus}$ (%) | | MA-STRIPS (%) | | MA-STRIPS$^{\ominus}$ (%) | |
|---|---|---|---|---|---|---|---|---|
| **Blocksworld** | 0 | 0 | 16 | 0 | 26 | 0 | 26 | 0 |
| **Depots** | 0 | 0 | 5 | 6 | 39 | 6 | 39 | 6 |
| **Driverlog** | 0 | 0 | 38 | 10 | 36 | 8 | 38 | 10 |
| **Elevators** | 33 | 18 | 33 | 18 | 74 | 37 | 74 | 37 |
| **Logistics** | 9 | 10 | 9 | 10 | 70 | 32 | 70 | 32 |
| **Openstacks** | 0 | 0 | 34 | 0 | 17 | 0 | 34 | 0 |
| **Rovers** | 70 | 54 | 70 | 54 | 77 | 59 | 77 | 59 |
| **Satellite** | 19 | 3 | 46 | 79 | 48 | 76 | 48 | 79 |
| **Woodworking** | 3 | 0 | 4 | 0 | 29 | 2 | 29 | 2 |
| **Zenotravel** | 19 | 8 | 35 | 83 | 60 | 77 | 61 | 83 |

Values are averages over problems in each domain

**Table 4** Number of MA-STRIPS problems solved by the compared planners: RDFF, GPPP, and PSM-RV

| **Domain** | RDFF | GPPP | PSM-RV |
|---|---|---|---|
| **Blocksworld** (34) | 6.8 | 3 | **26** |
| **Depots** (20) | 6.2 | **8** | 6 |
| **Driverlog** (20) | 14 | 9 | **15** |
| **Elevators** (30) | 2.9 | 16[a] | **30** |
| **Logistics** (20) | 5.8 | **20** | **20** |
| **Openstacks** (30) | 11.7 | 0[b] | **30** |
| **Rovers** (20) | 14.7 | 10 | **18** |
| **Satellite** (20) | 10.8 | **16** | 11 |
| **Woodworking** (30) | 5.6 | 0[b] | **24** |
| **Zenotravel** (20) | 6.1 | **20** | 13 |
| **Total** (244) | 84.6 | 102 | **193** |

[a] Used version of the domain in GPPP experiments without action costs, consisting of 16 problems
[b] GPPP does not support action costs

rithms RDFF and GPPP. Both planners, similarly to FMAP, are state-space search planners. RDFF [32] is based on distribution of A* algorithm with distributed heuristics with a variable number of agents involved in heuristic estimation. Greedy Privacy Preserving Planner (GPPP) [26] is based on an iterative deepening search in relaxed subproblems enhanced with landmarks. Table 4 shows that PSM-RV outperforms both state-of-the-art algorithms,[8] even though GPPP solved more instances of *Depots*, *Satellite*, and *Zenotravel* domains.

Table 5 shows overall results of PSM-RV algorithm for all privacy settings. Unfortunately, we are not aware of any planner that could be used to compare the performance on problems with internal goals. We can see that in some domains, increased privacy improves the performance of the planner, while in others, the performance decreases. For example, in *Satellites* problems, the most difficult goal is the final position pointing of a satellite (predicate `pointing`). If this goal is internal, then the complexity of the problem decreases. The opposite case is represented by *Openstacks* domain. When the goals are public, a producer (agent *manufacturer*) can more easily create a plan fitting the requirements of a consumer (agent

[8] We would like to thank Michal Štolba for evaluating RDFF and GPPP on our benchmarks.

**Table 5** Problem coverage of PSM-RV on benchmark domains with different privacy classifications

| **Domain** | FMAP | FMAP$^{\ominus}$ | MA-STRIPS | MA-STRIPS$^{\ominus}$ |
|---|---|---|---|---|
| **Blocksworld** (34) | **26** | 24 | **26** | **26** |
| **Depots** (20) | 3 | 4 | **6** | 4 |
| **Driverlog** (20) | 14 | **15** | **15** | **15** |
| **Elevators** (30) | **30** | **30** | **30** | **30** |
| **Logistics** (20) | **20** | **20** | **20** | **20** |
| **Openstacks** (30) | **30** | 12 | **30** | 26 |
| **Rovers** (20) | 16 | 14 | **18** | **18** |
| **Satellite** (20) | 9 | **18** | 11 | 17 |
| **Woodworking** (30) | **27** | 22 | 24 | 23 |
| **Zenotravel** (20) | **18** | 16 | 13 | 16 |
| **Total** (244) | 193 | 175 | 193 | **195** |

Number of problems in each domain is in parenthesis

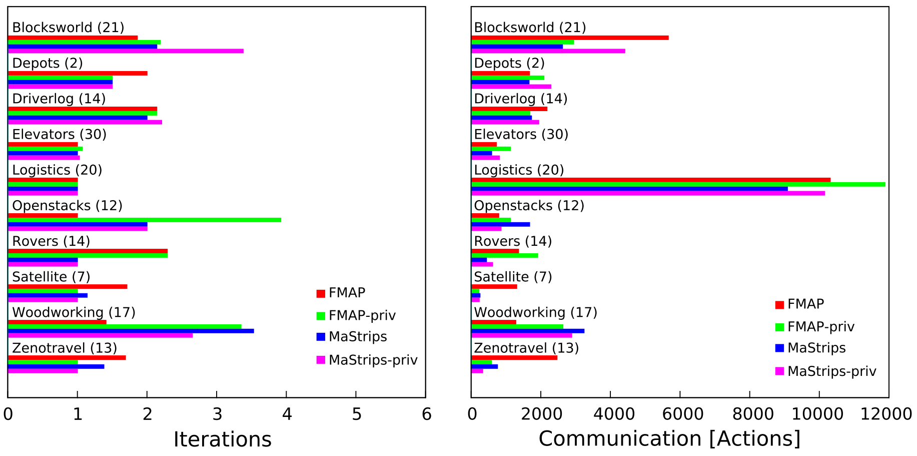

**Fig. 11** Performance of PSM-RV with different privacy classifications measured by the number of iterations and amount of communication. Values are averages over those problems solved by all the planners. Number of the problems considered for each domain is in parenthesis

*manager*). Internal goals improve the performance with MA-STRIPS classifications, but the effect on FMAP classifications is the opposite. The best overall coverage is for MA-STRIPS$^{\ominus}$ with 195 solved problems out of 244 problems. This shows that increase of internal facts does not only support privacy, but it can also improve performance (Fig. 11).

## 10 Conclusions

We have proposed a sound and complete approach for privacy preserving multiagent planning with external actions. It combines compilation for a classical planner with a compact representation of local plans. A compact representation of local plans in the form planning state machines (PSM) allows us to effectively implement desired operations, namely PSM intersection and public projection. Improving the planner with distributed delete-relaxation heuristics (PSM-R) and approximative plan analysis (PSM-V) proved to be effective in practice. Each of the improvements helps to solve a particular class of problems, and we have shown that both the improvements can be combined together without impairing each other (PSM-RV). Experimental evaluation revealed that PSM-RV outperforms state-of-the-art planner FMAP by more than 8 %. The improvements also reduced required communication by decreasing the number of iterations needed to solve benchmark problems. Private goals were added to the planner by a compilation scheme providing transformation of private goals to public ones without loss of privacy.

We have performed analysis of common multiagent planning benchmarks from the perspective of different privacy classifications. An indisputable advantage of our approach is that it can be used under different privacy classifications which has been confirmed by experiments. The result is that PSM-RV is best performing on the privacy classification defined by MA-STRIPS with private goals, that is, on the classification which contains the lowest amount of public facts out of the tested classifications. Usability of our planner with different privacy classifications allowed us to directly compare our approach with other planners designed particularly for MA-STRIPS privacy classification with public goals. The result is that PSM-RV outperforms state-of-the-art planners RDFF and GPPP.

The proposed planning approach constitutes a generic scheme which can be extended with other improvements and heuristics. While two such extensions were presented in this work, other extensions can be introduced to target most recent challenges in multiagent planning. The extensions targeting planning with a pair-wise visibility of facts, planning in a lifted form, and planning with joint actions are left for future work. With these extensions, we believe the presented approach becomes a scalable multiagent planner necessary to tackle real-world problems.

**Acknowledgments** This research was supported by the Czech Science Foundation (Grant No. 13-22125S).

## Appendix: Proofs of main theorems

**Theorem** 1 *Public plan $\sigma$ of $\Pi$ is extensible if and only if $\sigma$ is $\alpha$-extensible for every agent $\alpha$.*

*Proof* ($\Rightarrow$). When $\sigma$ is extensible, there is $\pi \in \mathsf{sols}(\Pi)$ such that $\pi \rhd \star = \sigma$. Let $\alpha$ be arbitrary but fixed. Let us construct plan $\pi_\alpha$ of $\Pi \rhd \alpha$ from $\pi$ by removing internal actions of agents other than $\alpha$ and by applying projection to the remaining actions obtaining $\pi_\alpha = \langle a \rhd \alpha : a \in \pi \text{ and } a \in \mathsf{pub\text{-}actions}(\Pi) \cup \mathsf{int\text{-}actions}(\alpha)\rangle$. Clearly $\pi_\alpha \rhd \star = \sigma$ because $\pi_\alpha$ preserves the order of public actions. To prove $\pi_\alpha \in \mathsf{sols}(\Pi \rhd \alpha)$, we first observe that no action $b$ internal for $\beta \neq \alpha$ can change state $s$ of $\Pi$ in a way observable by $\alpha$, that is, $\gamma(s, b) \rhd \alpha = s \rhd \alpha$. Hence, the sequence of states (of $\Pi$) which proves $\pi \in \mathsf{sols}(\Pi)$ can be easily transformed to a sequence of states of $(\Pi \rhd \alpha)$ which proves $\pi_\alpha \in \mathsf{sols}(\Pi \rhd \alpha)$. Thus $\sigma$ is $\alpha$-extensible. ($\Leftarrow$) For every agent $\alpha_i$, $\sigma$ is $\alpha_i$-extensible and thus there is some local solution $\pi_i$ such that $\pi_i \in \mathsf{sols}(\Pi \rhd \alpha_i)$ and $\pi_i \rhd \star = \sigma$. When more than one local solutions exist, we can choose an arbitrary from them. Now we construct a solution $\pi$ of $\Pi$ from local solutions $\pi_i$'s as follows. We split each $\pi_i$ at the positions of public actions from $\sigma$, and we join the corresponding internal parts of different plans together. Internal actions of different agents cannot interact through a shared fact (otherwise this fact would be public and these actions would be public too), and thus, we can join different internal parts in any order, preserving only the order of actions of individual agents.

Then we construct $\pi$ of $\Pi$ from $\sigma$ by translating ids from $\sigma$ to corresponding actions of $\Pi$ and by adding the joined parts between corresponding public actions in $\sigma$.

Clearly $\pi \triangleright \star = \sigma$ and $\pi \in \mathsf{sols}(\Pi)$. Hence, $\sigma$ is extensible. □

**Lemma** 3 *Let $\Pi$ be a classical* STRIPS *problem, let $\Gamma$ be a PSM of $\Pi$, and let $\pi \in \mathsf{sols}(\Pi)$. Then $\Gamma \oplus \pi$ is correctly defined and* $\mathsf{accept}(\Gamma) \cup \{\pi\} \subseteq \mathsf{accept}(\Gamma \oplus \pi)$.

*Proof* Let us prove that $\Gamma \oplus \pi$ is correctly defined as specified by Definition 5. Properties (1)–(3) are trivial. Property (4) is satisfied because $\pi \in \mathsf{sols}(\Pi)$ and hence $G \subseteq s_n$ where $s_n$ is the last state from Definition 7. It follows from Definition 7 that both PSMs are defined on the same alphabet and that $\Gamma$ is a sub-automaton of $\Gamma \oplus \pi$. Hence, clearly $\mathsf{accept}(\Gamma) \subseteq \mathsf{accept}(\Gamma \oplus \pi)$. Moreover, the sequence of states $s_0, \ldots, s_n$ from Definition 7 proves that $\pi \in \mathsf{accept}(\Gamma \oplus \pi)$. Hence, the claim. □

**Lemma** 4 *Let $\Pi \triangleright \alpha$ be a local problem of agent $\alpha$ and let $\Gamma$ be a PSM of $\Pi \triangleright \alpha$. Then $\Gamma \triangleright \star$ is a public PSM of $\Pi$ and* $\mathsf{accept}(\Gamma \triangleright \star) = \{\pi \triangleright \star : \pi \in \mathsf{accept}(\Gamma)\}$.

*Proof* Let $\Delta = \Gamma \triangleright \star$. Let $\delta_\Gamma$ be the transition function of $\Gamma$ and let $\delta_\Delta$ be the transition function of $\Delta$. Let us prove the inclusions ($\subseteq$) and ($\supseteq$) separately.

($\subseteq$) Let $\sigma \in \mathsf{accept}(\Delta)$ and let $\sigma = \langle id_1, \ldots, id_n \rangle$. Let $s_0, \ldots, s_n$ be the sequence of states of $\Delta$ which proves $\sigma \in \mathsf{accept}(\Delta)$. Now we can sequentially process these actions and construct a sequence of action ids $\pi'$ such that $\pi' \in \mathsf{accept}(\Gamma)$ and $\pi' \triangleright \star = \sigma$ as follows. Thanks to the integer labels, we can unambiguously translate every state of $\Delta$ to the state of $\Gamma$ using function $\rho^{-1}$. Let $t_i = \rho^{-1}(s_i)$ for $0 \le i \le n$. We start with empty $\pi'$. We know that the transition from $s_{i-1}$ to $s_i$ labeled by $id_i$ has been added to $\Delta$ by line 17 of Algorithm 3. Hence, there is state $r$ of $\Gamma$ such that $r \in \mathsf{int\text{-}closure}(t_{i-1})$ and $\delta_\Gamma(r, id_i) = t_i$. Hence, there has to be a (possibly empty) sequence of internal action ids $\langle id'_1, \ldots, id'_l \rangle$ which proves $r \in \mathsf{int\text{-}closure}(t_{i-1})$. We simply append $\langle id'_1, \ldots, id'_l, id_i \rangle$ to $\pi'$. In this way, we construct $\pi'$ by sequential processing of all action ids from $\sigma$. We know that $s_0$ is the initial state of $\Delta$ and also that $t_0$ is the initial state of $\Gamma$. It holds that $\pi' \triangleright \star = \sigma$ because all the actions from $\sigma$ were added to $\pi'$ in the right order and the additionally added actions are internal. To prove the claim, it is now enough to check that $\pi' \in \mathsf{accept}(\Gamma)$. When $t_n$ is an accepting state of $\Gamma$, we are done. Otherwise, $s_n$ is marked as an accepting state of $\Delta$ by line 19 and therefore there exists some accepting state $r'$ of $\Gamma$ such that $t_n \in \mathsf{int\text{-}closure}(r')$. Finally, we append internal action ids which prove $t_n \in \mathsf{int\text{-}closure}(r')$ to $\pi'$. Thus $\pi' \in \mathsf{accept}(\Gamma)$ and hence the claim.

($\supseteq$) Let $\pi \in \mathsf{accept}(\Gamma)$ and let $\sigma = \pi \triangleright \star$. We simulate the plan $\pi = \langle id_1, \ldots, id_n \rangle$ in the state space of $\Gamma$. We shall show that this simulation directly corresponds to the simulation of $\sigma$ in $\Delta$. Let $s_0, \ldots, s_n$ be the sequence of states of $\Gamma$ which proves $\pi \in \mathsf{accept}(\Gamma)$. Clearly the initial state of $\Gamma$ (that is, $s_0$) is translated by $\rho$ to the initial state of $\Delta$. For a transition from $s_{i-1}$ to $s_i$ labeled by $id_i$ in $\Gamma$, we distinguish two following two cases. Either (1) $id_i$ is public or (2) internal. If $id_i$ is public, it is trivially added by line 17 to $\Delta$ and thus we can follow the corresponding transition in $\Delta$. If $id_i$ is internal, we find the first transition with public action $\delta(s_{j-1}, id_j) \to s_j$, $j > i$. Note that all internal actions are removed when doing a public projection. Thus, we can proceed similarly to the previous case having virtual transition from $\delta(s_{i-1}, id_i) \to s_j$ with the only difference that now the needed transition is added because $s_j \in \mathsf{int\text{-}closure}(s)$. It can happen that no such index $j$ exists, i.e., the plan $\pi$ ends with a sequence of internal actions. In that case, the state $\rho(s_{i-1})$ is going to be added to the goal states at line 19. □

**Lemma** 5 *The intersection* $\Delta_1 \cap \Delta_2$ *of two public PSMs* $\Delta_1$ *and* $\Delta_2$ *of* $\Pi$ *is a correctly defined public PSM of* $\Pi$ *and the following holds.*

$$\mathsf{accept}(\Delta_1 \cap \Delta_2) = \mathsf{accept}(\Delta_1) \cap \mathsf{accept}(\Delta_2)$$

*Proof* Let $\Delta_1 = \langle \Sigma, S_1, I, \delta_1, F_1 \rangle$ and $\Delta_2 = \langle \Sigma, S_2, I, \delta_2, F_2 \rangle$. Let $\Delta_0 = \Delta_1 \cap \Delta_2$. Let us first prove that the $\Delta_1 \cap \Delta_2$ is a correctly defined PSM of MA-STRIPS problem $\Pi$ as specified in Definition 9. Properties (1) to (3) are trivially fulfilled. Property (4) is proved by Definition 12 (2) and property (5) by Definition 12 (3). Now let us prove the inclusions ($\subseteq$) and ($\supseteq$) separately.

($\subseteq$) Let $\sigma = \langle id_1, \ldots, id_n \rangle$ be a public plan such that $\sigma \in \mathsf{accept}(\Delta_0)$. Let $\langle s_0, l_0 \rangle, \ldots, \langle s_n, l_n \rangle$ be the sequence of states which proves $\sigma \in \mathsf{accept}(\Delta_0)$. Thanks to the distinctiveness property of an injective function $\cdot$, we can find $i_k$ and $j_k$ such that $l_k = i_k \cdot j_k$ for every $0 \leq k \leq n$. It holds that $\langle s_n, i_n \rangle \in F_1$ and $\langle s_n, j_n \rangle \in F_2$ by Definition 12 (3). Hence, the sequence of states $\langle s_0, i_0 \rangle, \ldots, \langle s_n, i_n \rangle$ proves that $\sigma \in \mathsf{accept}(\Delta_1)$ by Definition 12 (2). Similarly, $\sigma \in \mathsf{accept}(\Delta_2)$ and hence the claim.

($\supseteq$) Let $\Delta_0 = \langle \Sigma, S_0, I, \delta_1, F_0 \rangle$. Let $\sigma = \langle id_1, \ldots, id_n \rangle$ be a public plan such that $\sigma \in \mathsf{accept}(\Delta_1) \cap \mathsf{accept}(\Delta_2)$ Let $\langle s_0, i_0 \rangle, \ldots, \langle s_n, i_n \rangle$ be the sequence of states which proves $\sigma \in \mathsf{accept}(\Delta_1)$ and let $\langle q_0, j_0 \rangle, \ldots, \langle q_n, j_n \rangle$ be the sequence of states which proves $\sigma \in \mathsf{accept}(\Delta_2)$. We know that $\langle s_0, i_0 \rangle = \langle q_0, j_0 \rangle = I$. Hence, it is easy to check by Definition 9 (4) that $s_k = q_k$ for all $0 \leq k \leq n$. Also we know that $\langle s_n, i_n \rangle \in F_1$ and $\langle s_n, j_n \rangle = \langle q_n, j_n \rangle \in F_2$. Hence, $\langle s_n, i_n \cdot j_n \rangle \in F_0$ by Definition 12 (3). Now the sequence of states $\langle s_0, i_0 \cdot j_0 \rangle, \ldots, \langle s_n, i_n \cdot j_n \rangle$ proves that $\sigma \in \mathsf{accept}(\Delta_0)$ by Definition 12 (2) and hence the claim. □

**Theorem** 2 *Let* $\Pi$ *be an* MA-STRIPS *problem and* $\Delta = \texttt{PsmPlanComplete}(\Pi)$. *It holds that* $\mathsf{accept}(\Delta) = \{\pi \triangleright \star : \pi \in \mathsf{sols}(\Pi)\}$.

*Proof* Let $\Gamma_\alpha$ denote the complete PSM of $\Pi \triangleright \alpha$ and $\Delta_\alpha = \Gamma_\alpha \triangleright \star$. Let us prove the inclusions ($\subseteq$) and ($\supseteq$) separately.

($\subseteq$) Let $\sigma \in \mathsf{accept}(\Delta)$. Then $\sigma \in \mathsf{accept}(\Delta_\alpha)$ for every $\alpha$ by Lemma 5. Hence, for every $\alpha$ there is $\pi_\alpha \in \mathsf{accept}(\Gamma_\alpha)$ such that $\sigma = \pi_\alpha \triangleright \star$ by Lemma 4. By Lemma 2 $\pi_\alpha \in \mathsf{sols}(\Pi \triangleright \alpha)$ and thus $\sigma$ is $\alpha$-extensible for every $\alpha$. Hence, $\sigma$ is extensible by Theorem 1.

($\supseteq$) Let $\pi \in \mathsf{sols}(\Pi)$ and $\sigma = \pi \triangleright \star$. Hence, $\sigma$ is extensible and thus also $\alpha$-extensible for every $\alpha$ by Theorem 1. Hence, for every $\alpha$ there is $\pi_\alpha \in \mathsf{sols}(\Pi \triangleright \alpha)$ with $\pi_\alpha \triangleright \star = \sigma$. Clearly $\pi_\alpha \in \mathsf{accept}(\Gamma_\alpha)$ by Lemma 2 and thus $\sigma \in \mathsf{accept}(\Delta_\alpha)$ by Lemma 4. Thus $\sigma \in \mathsf{accept}(\Delta_\alpha)$ for all $\alpha$ and hence the claim by Lemma 5. □

**Theorem** 4 *Let* $\Pi$ *be an* MA-STRIPS *problem, let* $\alpha$ *be an agent of* $\Pi$, *and let* $\sigma$ *be a public plan of* $\Pi$. *Then* $\sigma$ *is* $\alpha$*-extensible iff* $\mathsf{sols}(\Pi \circledast_\alpha \sigma) \neq \emptyset$.

*Proof* ($\Rightarrow$) Let $\sigma$ be $\alpha$-extensible. Hence there is $\pi \in \mathsf{sols}(\Pi \triangleright \alpha)$ such that $\pi \triangleright \star = \sigma$. Clearly $\pi$ contains only public actions in the order given by $\sigma$. The rest are internal actions of $\alpha$. We construct $\pi_0$ from $\pi$ by replacing public actions by their respective landmark actions. It is easy to verify that $\pi_0 \in \mathsf{sols}(\Pi \circledast_\alpha \sigma)$.

($\Leftarrow$) Let $\pi \in \mathsf{sols}(\Pi \circledast_\alpha \sigma)$. Clearly $\pi \triangleright \star = \sigma$. Let us construct $\pi_0$ be translating landmark actions back to their original actions. It still holds $\pi_0 \triangleright \star = \sigma$, and it is easy to check that $\pi_0 \in \mathsf{sols}(\Pi \triangleright \alpha)$. Hence, $\sigma$ is $\alpha$-extensible. □

## References

1. Amir E, Engelhardt B (2003) Factored planning. In: Gottlob G, Walsh T (eds) IJCAI-03, Proceedings of the eighteenth international joint conference on artificial intelligence, Acapulco, Mexico, 9–15 Aug 2003. Morgan Kaufmann, pp 929–935
2. Bacchus F, Yang Q (1994) Downward refinement and the efficiency of hierarchical problem solving. Artif Intell 71(1):43–100
3. Bhattacharya S, Kumar V, Likhachev M (2010) Search-based path planning with homotopy class constraints. In: Felner A, Sturtevant NR (eds) SOCS. AAAI Press, Menlo Park
4. Borrajo D (2013) Plan sharing for multi-agent planning. In: Proceedings of DMAP workshop of ICAPS'13, pp 57–65
5. Boutilier C, Brafman RI (2001) Partial order planning with concurrent interacting actions. Artif Intell 14:105–136
6. Brafman R, Domshlak C (2008) From one to many: planning for loosely coupled multi-agent systems. In: Proceedings of ICAPS'08, vol 8, pp 28–35
7. Brafman RI, Domshlak C (2006) Factored planning: how, when, and when not. In: Proceedings, the twenty-first national conference on artificial intelligence and the eighteenth innovative applications of artificial intelligence conference, 16–20 July 2006, Boston, Massachusetts, USA. AAAI Press, pp 809–814
8. Brenner M (2003) Multiagent planning with partially ordered temporal plans. In: Proceedings of the 18th international joint conference on artificial intelligence, IJCAI'03. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp 1513–1514
9. Darwiche A, Hopkins M (2001) Using recursive decomposition to construct elimination orders, jointrees, and dtrees. In: Benferhat S, Besnard P (eds) Symbolic and quantitative approaches to reasoning with uncertainty, 6th European conference, ECSQARU 2001, Toulouse, France, 19–21 Sept 2001, Proceedings, Lecture Notes in Computer Science, vol 2143. Springer, pp 180–191. doi:10.1007/3-540-44652-417
10. Dimopoulos Y, Hashmi MA, Moraitis P (2010) Extending SATPLAN to multiple agents. In: Proceedings of the 30th international conference on innovative techniques and applications of artificial intelligence (SGAI'10), pp 137–150
11. Durkota K, Komenda A (2013) Deterministic multiagent planning techniques: experimental comparison (short paper). In: Proceedings of DMAP workshop of ICAPS'13, pp 43–47
12. Fabre E, Jezequel L, Haslum P, Thiébaux S (2010) Cost-optimal factored planning: promises and pitfalls. In: Brafman RI, Geffner H, Hoffmann J, Kautz HA (eds) ICAPS. AAAI, pp 65–72
13. Fikes R, Nilsson N (1971) STRIPS: a new approach to the application of theorem proving to problem solving. In: Proceedings of the 2nd international joint conference on artificial intelligence, pp 608–620
14. Ghallab M, Nau DS, Traverso P (2004) Automated planning: theory and practice. Elsevier, Amsterdam
15. Hoffmann J, Nebel B (2001) The FF planning system: fast plan generation through heuristic search. J Artif Intell Res (JAIR) 14:253–302
16. Hopcroft JE, Motwani R, Ullman JD (2006) Introduction to automata theory, languages, and computation, 3rd edn. Addison-Wesley Longman Publishing Co. Inc., Boston
17. Jakubův J, Tožička J, Komenda A (2015) Multiagent planning by plan set intersection and plan verification. In: Proceedings ICAART'15
18. Jakubův J, Wells JB (2010) Expressiveness of generic process shape types. In: TGC'10, LNCS, vol 6084. Springer, pp 103–119
19. Jensen RM, Veloso MM (2000) OBDD-based universal planning for synchronized agents in non-deterministic domains. J Artif Intell Res 13:189–226
20. Jezequel L, Fabre E (2012) A#: a distributed version of A* for factored planning. In: Proceedings of the 51th IEEE conference on decision and control (CDC'12), pp 7377–7382
21. Jonsson A, Rovatsos M (2011) Scaling up multiagent planning: a best-response approach. In: Proceedings of the 21st international conference on automated planning and scheduling (ICAPS'11), pp 114–121
22. Kelareva E, Buffet O, Huang J, Thiébaux S (2007) Factored planning using decomposition trees. In: Veloso MM (ed) IJCAI 2007, Proceedings of the 20th international joint conference on artificial intelligence, Hyderabad, India, 6–12 Jan 2007, pp 1942–1947
23. Kovacs DL (2012) A multi-agent extension of pddl3.1. In: Proceedings of the 3rd workshop on the international planning competition (IPC), 22nd international conference on automated planning and scheduling (ICAPS-2012). Atibaia, Sao Paulo, Brazil, pp 19–27
24. Lansky AL, Getoor L (1995) Scope and abstraction: two criteria for localized planning. In: Proceedings of the fourteenth international joint conference on artificial intelligence, IJCAI 95, Montréal Québec, Canada, 20–25 Aug 1995, vol 2. Morgan Kaufmann, pp 1612–1619

25. Makholm H, Wells JB (2005) Instant polymorphic type systems for mobile process calculi: just add reduction rules and close. In: ESOP'05, LNCS, vol 3444. Springer, pp 389–407
26. Maliah S, Shani G, Stern R (2014) Privacy preserving landmark detection. In: Proceedings of ECAI'14
27. Nissim R, Brafman RI (2012) Multi-agent A* for parallel and distributed systems. In: Proceedings of AAMAS'12, AAMAS '12. Richland, SC, pp 1265–1266
28. Nissim R, Brafman RI (2013) Cost-optimal planning by self-interested agents. In: Proceedings of DMAP workshop of ICAPS'13, pp 1–7
29. Oglietti M, Cesta A (2004) CSTRIPS: towards explicit concurrent planning. In: Proceedings of ninth national symposium of 'Associazione Italiana per l'Intelligenza Artificiale AI*IA'04, third Italian workshop on planning and scheduling IWP'04. Perugia, Italy. ISBN: 88-89422-09-2
30. Pellier D (2010) Distributed planning through graph merging. In: Proceedings of the 2nd international conference on agents and artificial intelligence (ICAART 2010), pp 128–134
31. Štolba M, Komenda A (2013) Fast-forward heuristic for multiagent planning. In: Proceedings of DMAP workshop of ICAPS'13, vol 120, pp 75–83
32. Štolba M, Komenda A (2014) Relaxation heuristics for multiagent planning. In: 24th International conference on automated planning and scheduling (ICAPS), pp 298–306
33. Torreño A, Onaindia E, Sapena O (2014) FMAP: distributed cooperative multi-agent planning. Appl Intell 41(2):606–626
34. Torreño A, Onaindia E, Sapena O (2014) A flexible coupling approach to multi-agent planning under incomplete information. Knowl Inf Syst 38(1):141–178
35. Tožička J, Jakubův J, Durkota K, Komenda A, Pěchouček M (2014) Multiagent planning supported by plan diversity metrics and landmark actions. In: Proceedings ICAART'14
36. Tožička J, Jakubův J, Komenda A (2014) Generating multi-agent plans by distributed intersection of finite state machines. In: Proceedings ECAI'14, pp 1111–1112

**Jan Tožička** is a researcher of Agent Technology Center (ATG) at the Faculty of Electrical Engineering (FEE), Czech Technical University in Prague (CTU) since 2010. He concentrates on multiagent and distributed planning. Previously, Jan worked on distributed learning algorithms and on the diverse planning. He holds master degree and doctorate in natural sciences in Computer Science from Faculty of Mathematics and Physics at Charles University in Prague. Prior to this position, Jan worked as a software engineer in IndigoVision Ltd. in Edinburgh, where he focused on design and implementation of integration modules and network video recorders.

**Jan Jakubův** is a postdoctoral researcher at Agent Technology Center (ATG) at Department of Computer Science, FEE, Czech Technical University in Prague. Jan earned his M.Sc. degree at Charles University in Prague in 2006 under the supervision of Prof. Petr Štěpánek. In 2007, Jan received an EPSRC scholarship to undertake a Ph.D. study at Heriot-Watt University in Edinburgh under the supervision of Dr. Joe Wells and Prof. Fairouz Kamareddine. His Ph.D. thesis was successfully defended with no corrections in 2010. His research interests include automated reasoning, automated planning, multiagent systems, process calculi, and type systems.

**Antonín Komenda** is a research fellow at the Agent Technology Center (ATG) at the Faculty of Electrical Engineering (FEE), Czech Technical University in Prague (CTU). His research focuses on domain-independent multiagent planning. He focuses on both deterministic and uncertain planning models. Antonín's project work is mostly related to research in planning and applied mission and tactical planning. Antonín holds Ph.D. from FEE, CTU and was a postdoctoral fellow at Technion Israel Institute of Technology in 2013–2014.

**Michal Pěchouček** is a professor at the Czech Technical University in Prague, being the Head of Agent Technology Center. Furthermore, he is also the Deputy head for research at Department of Computer Science and the Director of Open Informatics Programme. In 2011, he was a visiting professor at University of Southern California, TEAMCORE lab (Fulbright fellow), in 2006 visiting scientist at University of Edinburgh (Royal Society of Edinburgh), in 2003 visiting professor at State University of New York at Binghamton, and in 2000 postdoc researcher at University of Calgary. He is a chair of the board of directors for EURAMAS, member of the board of directors of IFAAMAS, visiting/honorary member of the Artificial Intelligence Application Institute, University of Edinburgh, and member of the advisory board of the Center for Advanced Information Technologies, SUNY Binghamton.