Multiagent Planning by Plan Set Intersection and Plan Verification

Jan Jakubův, Jan Tožička and Antonín Komenda
Agent Technology Center, Department of Computer Science, Czech Technical University, Prague, Czech Republic

Keywords: Multiagent Planning, Action Landmarks, Plan Verification, Process Calculi, Type Systems, Delete Relaxation.

Abstract: Multiagent planning is a coordination technique used for deliberative acting of a team of agents. One of vital planning techniques uses declarative description of agents' plans based on Finite State Machines and their later coordination by intersection of such machines with successive verification of the resulting joint plans. In this work, we firstly propose to use projections of agents' actions directly for multiagent planning based on iterative building of a coordinated multiagent plan. Secondly, we describe integration of the static analysis provided by process calculi type systems for approximate verification of exchanged local plans. Finally, we compare our approach with current state-of-the-art planner on an extensive benchmark set.

1 INTRODUCTION

Intelligent agents requested to act together in a team require to some extent an ability to plan their actions in advance. If the agents prepare complete plans towards their goals, the problem they have to solve is a form of multiagent planning.

Similarly to classical planning, our multiagent planning approach assumes STRIPS (Fikes and Nilsson, 1971) actions, which are deterministic and described by precondition and effects on the environment they are executed in. Thereby, the action state progression follows the STRIPS principles as well.

Although the action model is STRIPS, the complete multiagent planning model is subsequent to a recent extension of STRIPS by Brafman & Domshlak called MA-STRIPS (Brafman and Domshlak, 2008). In MA-STRIPS, the agents are cooperative with common goals and the resulting multiagent plan prescribes their coordinated acting from the initial state of the environment towards the goals. The agents are heterogeneous with different capabilities described by their STRIPS actions. Straightforwardly, their actions define parts of the environment they can affect and this gives rise to their local planning problems. Conveniently, this (partial) "separation of concerns" helps to increase efficiency of the planning process and provides intrinsic separation of public information the agents have to share and internal facts, which can be kept private.

The multiagent planning approach proposed in this work extends recent works by Tožička, et al. on representation of multiagent plans in form of Finite State Machines and their merging (Tožička et al., 2014b) and plan generation using diverse planning with homotopy class constraints with testing of usability of partial plans among the agents by compilation into planning landmarks (Tožička et al., 2014a).

In this work, we initially propose to use a projections of actions (Nissim and Brafman, 2012) directly for multiagent planning, which was in the literature used so far only in relaxation heuristic estimations (Štolba and Komenda, 2014). The main improvement is based on integration of theory and analysis provided by process calculi and their type systems. Concretely, we use generic process calculi type system scheme POLY★ (Makholm and Wells, 2005; Jakubův and Wells, 2010) for approximate verification of foreign plans received from other agents, which prospectively increases efficiency of search for coordinated multiagent plans. Finally, we compare our approach with current state-of-the-art planner FMAP (Torreño et al., 2014) on an extensive benchmark set.

2 MULTIAGENT PLANNING

We consider a number of *cooperative* and *coordinated* agents featuring distinct sets of capabilities (actions) which 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

Jakubův J., Tožička J. and Komenda A..
Multiagent Planning by Plan Set Intersection and Plan Verification.
DOI: 10.5220/0005222101730182
In *Proceedings of the International Conference on Agents and Artificial Intelligence* (ICAART-2015), pages 173-182
ISBN: 978-989-758-074-1

the MA-STRIPS problem (Brafman and Domshlak, 2008) required for the following sections.

2.1 Planning Problem

An MA-STRIPS planning problem Π is defined as a quadruple $\Pi = \langle P, \{A_i\}_{i=1}^n, I, G\rangle$, where P is a set of facts, A_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 conditions on the goal states. Given Π, we use A to denote all the actions from Π, that is, $A = \bigcup_{i=1}^n A_i$.

An *action* an agent can perform is a triple of subsets of P which in turn denote the set of *preconditions*, the set of *add effects*, and the set of *delete effects*. Selector functions $\mathsf{pre}(a)$, $\mathsf{add}(a)$, and $\mathsf{del}(a)$ are defined so that $a = \langle \mathsf{pre}(a), \mathsf{add}(a), \mathsf{del}(a)\rangle$. Moreover let $\mathsf{eff}(a) = \mathsf{add}(a) \cup \mathsf{del}(a)$.

An *agent* is identified with its capabilities, that is, an agent $\alpha = A_i = \{a_1, \ldots, a_m\}$ is characterized by a finite repertoire of actions it can perform in the environment. We use metavariables α and β to range over agents from Π. A *planning state* s is a finite set of facts and we say that fact p holds in s iff $p \in s$. When $\mathsf{pre}(a) \subseteq s$ then *state progression* function γ is defined classically as $\gamma(s,a) = (s \setminus \mathsf{del}(a)) \cup \mathsf{add}(a)$.

2.2 Public and Internal Classification

In 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* α when it is not public but mentioned by some action of α. A fact is *relevant for* α when it is either public or internal for α. Relevant facts contain all the facts which agent α needs to understand, because other facts are internal for other agents and thus not directly concerns α. Given Π, the set pub of public facts, and sets $\mathsf{int}(\alpha)$ and $\mathsf{rel}(\alpha)$ of facts internal and relevant for α are formally defined as follows. Let $\mathsf{facts}(a) = \mathsf{pre}(a) \cup \mathsf{add}(a) \cup \mathsf{del}(a)$ and similarly $\mathsf{facts}(\alpha) = \bigcup_{a \in \alpha} \mathsf{facts}(a)$.

$$\begin{aligned} \mathsf{pub} &= \bigcup_{\alpha \neq \beta} (\mathsf{facts}(\alpha) \cap \mathsf{facts}(\beta)) \\ \mathsf{int}(\alpha) &= \mathsf{facts}(\alpha) \setminus \mathsf{pub} \\ \mathsf{rel}(\alpha) &= \mathsf{pub} \cup \mathsf{int}(\alpha) \end{aligned}$$

It is possible to extend the set of public facts to contain additionally some facts that would be internal by the above definition. It is common in literature (Nissim and Brafman, 2012) to require that all the goals are public. Then pub is defined as the minimal superset of the intersection from the definition that satisfies $G \subseteq \mathsf{pub}$. In the rest of this paper we suppose $G \subseteq \mathsf{pub}$ and also another simplification common in literature (Brafman and Domshlak, 2008) which says that A_i are pairwise disjoint[1].

MA-STRIPS further extends this classification of facts to actions as follows. An action is *public* when it has a public effect (that is, $\mathsf{eff}(a) \cap \mathsf{pub} \neq \emptyset$) , otherwise it is *internal*. Strictly speaking, MA-STRIPS defines an action as public whenever it mentions a public fact even in a precondition (that is, when $\mathsf{facts}(a) \cap \mathsf{pub} \neq \emptyset$). However, as our approach does not rely on synchronization on public preconditions, we can consider actions with only public preconditions as internal. For our approach it is enough to know that internal actions do not *modify* public state.

2.3 Local Planning Problems

In MA-STRIPS model, agent actions are supposed to manipulate a shared global state when executed. In our approach to multiagent planning, a *local planning problem* is constructed for every agent α. Each local planning problem for α is a classical STRIPS problem where agent α has its own internal copy of the global state and where each agent is equipped 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 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* an arbitrary set $F \subseteq P$ of facts to agent α is the restriction of F to the facts relevant for α, that is, $F \triangleright \alpha = F \cap \mathsf{rel}(\alpha)$. Projection removes from F facts not relevant for α and thus it represents F as understood by agent α. The *projection* $a \triangleright \alpha$ *of* action a to agent α removes from a facts not relevant for α, again representing a as seen by α.

$$a \triangleright \alpha = \langle \mathsf{pre}(a) \triangleright \alpha,\ \mathsf{add}(a) \triangleright \alpha,\ \mathsf{del}(a) \triangleright \alpha \rangle$$

Note that $a \triangleright \alpha = a$ when $a \in \alpha$. Hence projection to α alters only actions of other agents.

In the multiagent planning approach presented in this paper, every agent α is from the beginning equipped with projections of other agents public actions. These projections, which we call *external actions*, describe how agent α sees effects of public actions of other agents. Given Π, the set $\mathsf{ext}(\alpha)$ of ex-

[1] These two conditions rules out *private goals* and *joint actions*. Any MA-STRIPS problem which does not satisfy the two conditions can be translated to an equivalent problem which satisfies them. However, a solution that would take advantage of private goals and joint actions is left for future research.

ternal actions of agent α is defined as follows.

$$\mathsf{ext}(\alpha) \quad = \quad \{a \triangleright \alpha : a \text{ is a public action of } \beta \neq \alpha\}$$

Recall that A denotes the set of all the actions from Π. The set $A \triangleright \alpha$ contains actions of α plus external actions it is defined as follows.

$$A \triangleright \alpha \quad = \quad \alpha \cup \mathsf{ext}(\alpha)$$

Now it is easy to define a *local planning problem* $\Pi \triangleright \alpha$ of agent α also called *projection of* Π to α.

$$\Pi \triangleright \alpha = \langle P \triangleright \alpha,\ A \triangleright \alpha,\ I \triangleright \alpha,\ G \rangle$$

2.4 Plans and Extensibility

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

A *plan* π is a sequence of actions $\langle a_1, \ldots, a_k \rangle$. A plan π 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 Π is a plan π whose execution transforms the initial state I to subset of G. A *local solution* is a solution of a local planning problem. Let $\mathsf{sols}(\Pi)$ and $\mathsf{sols}(\Pi \triangleright \alpha)$ denote the set of all solutions of a given problem.

A *public plan* σ 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. In order to avoid confusions between public and external versions of the same action, we suppose that actions are annotated with unique *id*s which are preserved under projection. From now on we consider public plans to be sequences of public action *id*s.

Let operator $\cdot^\star$ construct a public plan from plan π, that is, let $\pi^\star$ be the sequence of all public action *id*s from π preserving their order. A public plan σ is called *extensible* iff it be can extended to a solution of Π by insertion of internal actions of any agent, that is, iff there is $\pi \in \mathsf{sols}(\Pi)$ such that $\pi^\star = \sigma$. A public plan is called α-*extensible* iff it can be extended to a local solution of $\Pi \triangleright \alpha$ by insertion of internal actions of α, that is, iff there is agent α's plan $\pi_\alpha \in \mathsf{sols}(\Pi \triangleright \alpha)$ such that $\pi_\alpha^\star = \sigma$.

The following proposition states the correctness of the used approach to multiagent planning. Its direct consequence is that to find a solution of Π it is enough to find a local solution $\pi_\alpha \in \mathsf{sols}(\Pi \triangleright \alpha)$ which is β-extensible for every other agent β. A constructive proof can be found in (Tožička et al., 2014a).

Proposition 1. *Let public plan σ of Π be given. Public plan σ is extensible if and only if σ is α-extensible for every agent α.*

3 PLANNING STATE MACHINES

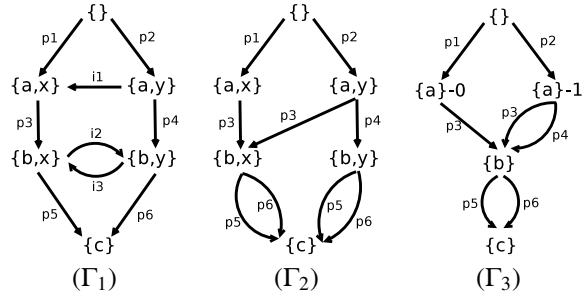
This section briefly restates the previous work by (Tožička et al., 2014b) our work is based on, while its extensions, which are the main contributions of this paper, are described in following sections. We use nondeterministic finite state machines (NFS) (Hopcroft et al., 2006) as a compact representation of a set of solutions of a STRIPS problem. We call an NFS that represents a set of solutions a *planning state machine* (PSM).

A *planning state machine (PSM)* of a STRIPS problem $\Pi = \langle P, A, I, G \rangle$ is a NFS $\Gamma = \langle A, S, I, \delta, F \rangle$ where the alphabet A is the set of actions of Π, states from S are subsets of P, the state transition function δ resembles the classical STRIPS state progression function γ, and $F \subseteq S$ contains all the states that satisfies goal G. To avoid confusions, we suppose that alphabet A contains unique action *id*s rather than full actions. The only requirement[2] on the state transition function δ is that $\delta(s,a) = \gamma(s,a)$. It is possible to construct a *complete* PSM that contains all possible states and transitions. A complete PSM of Π represents exactly all the solutions of Π. As the computation of a complete PSM can be inefficient, we also consider *partial* PSMs which represents only a subset of all solutions.

For every PSM Γ we can construct its public projection PSM $\Gamma^\star$ that represents public projections of the solutions. When Γ represents the set of solutions S we want $\Gamma^\star$ to represent exactly the set $\{\pi^\star : \pi \in S\}$. Once we have a PSM for every agent's local planning problem $\Pi \triangleright \alpha$, we can compute public projections of these PSMs and intersect them using a well-known intersection algorithm for NFS (Hopcroft et al., 2006). Any public solution σ in a non-empty intersection constitutes a public solution of the original MA-STRIPS problem Π. That is because σ is α-extensible for every α (as it comes from the intersection) and thus extensible by Proposition 1.

Figure 1 provides an example PSM Γ_1 demonstrating PSM public projection algorithm. First, PSM Γ_2 is obtained from the input Γ_1 by renaming internal actions to ε-transitions and eliminating them by the

[2] Although a PSM as defined here suggests a deterministic finite state machine, a non-deterministic transitions can be introduced by public projection defined later. Thus we prefer to work with non-deterministic machines from the beginning.



(Γ_1) (Γ_2) (Γ_3)

Figure 1: Example of computing PSM public projection (Tožička et al., 2014b). 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.

intersection algorithm (Hopcroft et al., 2006). Second, the public projection Γ_3 of Γ_1 is obtained from Γ_2 by projecting states (removing internal facts) and by unification of states with equal public projection. Hence $\Gamma_1^\star = \Gamma_3$. When two states with equal public projection differ in outgoing edges then they can not be unified and an integer *mark* is introduced to distinguish them. Note that if the two states `{a}-0` and `{a}-1` were unified in Γ_3, then the resulting PSM would also represent a public plan $\langle$`p1`, `p4`, `p5`$\rangle$ which does not correspond to any plan from Γ_1.

Algorithm 1 provides an overview of a distributed algorithm (Tožička et al., 2014b) to find a solution of an MA-STRIPS problem. We suppose that every agent α executes `PsmPlanDistributed` in a separate process, possibly on a separate machine. We suppose that agent processes can communicate with each other by sending structured messages. Every agent α starts with an empty PSM Γ_α which contains only the initial state of $\Pi \triangleright \alpha$. In every loop iteration, every agent generates a new plan of its local problem $\Pi \triangleright \alpha$ and it adds this plan to Γ_α. Then public projection $\Gamma_\alpha^\star$ is computed and exchanged with other agents. The easiest way to implement PSM projections exchange is when every agent sends its projection to every other agent. The projections can also be exchanged in a round-robin manner. The loop in the algorithm continues until the intersection $\bigcap_\beta(\Gamma_\beta^\star)$ is not empty. The operation written in ***bold italics*** is an optional extension described in Section 5.

By a new plan in the first step we mean a plan that was not generated in any of the previous iterations. To achieve this we have modified an existing planner FastDownward[3] so that it is able to generate a plan which differs from plans provided as an input. This extension is inspired by diverse planning with homotopy class constraints (Bhattacharya et al., 2010). Homotopy classes of plans are naturally defined by their

[3]http://www.fast-downward.org/

Algorithm 1: Distributed algorithm to find a solution of MA-STRIPS problem Π.

Function `PsmPlanDistributed`($\Pi \triangleright \alpha$) **is**
  $\Gamma_\alpha \leftarrow$ empty PSM (initial state of $\Pi \triangleright \alpha$);
  loop
    generate a new plan π_α of $\Pi \triangleright \alpha$;
    analyse plan π_α ***(Sec. 5)***;
    extend Γ_α with π_α;
    compute public projection $\Gamma_\alpha^\star$ of Γ_α;
    exchange PSM public projections;
    if intersection $\bigcap_\beta(\Gamma_\beta^{\text{pub}}) \neq \emptyset$ **then**
      return the intersection;
    end
    create landmarks from plans;
  end
end

public projections, that is, two plans belong to same homotopy class iff they have equal public projection.

In the last step of the loop, other agents plans are incorporated into the local planning problem $\Pi \triangleright \alpha$ using the principle of *prioritizing actions* and *soft-landmarks*. Prioritizing actions are implemented using action costs so that internal actions are preferred to public actions, and α's public actions are preferred to other agent actions. When agent α finds a new local solution it sends its public projection to all the other agents. Other agents then extend their local problem $\Pi \triangleright \beta$ to contain duplicated landmark actions from the received plan. These landmark actions have significantly decreased cost and they are interlinked using additional facts to ensure they are used in the order suggested by the public plan. See (Tožička et al., 2014a) for details.

4 PLANNING CALCULUS

In this section we show how classical planning can be expressed as a process calculus. Furthermore we show how to use existing process calculi type systems for static analysis of classical planning problems and how to use this static analysis for approximation of planning problem solvability. In the next section, this static analysis will be incorporated into the multiagent planning algorithm from the previous section (Algorithm 1).

4.1 Planning as Process Calculus

A typical *process calculus* is defined by a set of *processes* together with a binary *rewriting relation* ($\rightarrow$)

on these processes. Processes describe possible system states while the rewriting relation describes possible transitions between states. Hence $Q_0 \to Q_1$ means that the system can be transformed (in one step) from the state described by Q_0 to state Q_1.

Process calculi are usually used to model concurrent environments where several units (processes) engage in activity at the same time. Processes usually take form of programs and thus system states are identified with programs currently running in the system, while rewriting relation captures program evaluation.

Processes are usually constructed from atomic processes using standard operators. In this paper, we will use only the *parallel composition* operator ("$\mid$"). Process "$Q_0 \mid Q_1$" describes a system where processes Q_0 and Q_1 are running in parallel. We will also use standard *null* or *inactive* process denoted "$\mathsf{0}$". Parallel composition is considered commutative and associative with $\mathsf{0}$ being an identity element (that is, $Q \mid \mathsf{0} = Q$).

Given a STRIPS problem Π with set of facts P, we use facts $p \in P$ as atomic processes. Hence our processes correspond to planning states while the rewriting relation emulates action application by adjusting state (process) appropriately. The set of our processes is generated by the following grammar.

$$Q ::= \mathsf{0} \mid p \mid (Q_0 \mid Q_1)$$

That is, null process $\mathsf{0}$ is a process, every fact p is a process, and other processes can be constructed using parallel composition. Function $\lceil s \rceil$ encodes a state as a process.

$$\lceil \{p_1, p_2, \ldots, p_n\} \rceil = p_1 \mid p_2 \mid \cdots \mid p_n \mid \mathsf{0}$$

Furthermore, to simplify the presentation we consider processes to be equal modulo fact duplications (that is, "$(p \mid p) = p$" and so on).

The rewriting relation $\to$ is the minimal relation which satisfies the following rules.

$$\frac{a \in A}{\lceil \mathsf{pre}(a) \rceil \to \lceil \gamma(\mathsf{pre}(a), a) \rceil} \qquad \frac{Q_0 \to Q_1}{Q_0 \mid Q_2 \to Q_1 \mid Q_2}$$

Recall that the state progression function γ for action a applied to the state $\mathsf{pre}(a)$ is defined as follows.

$$\gamma(\mathsf{pre}(a), a) = (\mathsf{pre}(a) \setminus \mathsf{del}(a)) \cup \mathsf{add}(a)$$

Hence the first rule says that, for every action a, the process $\lceil \mathsf{pre}(a) \rceil$ can be rewritten to the process $\lceil \mathsf{pre}(a) \rceil$ with delete effects removed and add effects added. The second rule is a context rule which allows us to apply rewriting in the presence of additional facts. The following formally states that the rewriting relation $\to$ correctly captures planning action execution.

Proposition 2. *Let* $\Pi = \langle P, A, I, G \rangle$ *be a classical* STRIPS *problem such that* $\mathsf{del}(a) \subseteq \mathsf{pre}(a)$ *for every action* $a \in A$. *For any two planning states* s_0 *and* s_1 *the following holds.*

$$\lceil s_0 \rceil \to \lceil s_1 \rceil \quad \textit{iff} \quad \exists a \in A : \gamma(s_0, a) = s_1$$

The requirement that every action deletes only facts mentioned in its preconditions ($\mathsf{del}(a) \subseteq \mathsf{pre}(a)$) is necessary because the right-hand side of a rewriting rule can refer only to processes mentioned on the left-hand side. This requirement simplifies formal presentation while it does not restrict usability because problems not fulfilling the requirement can be translated[4] to equivalent problems which do so. Moreover, it is usually satisfied in practice.

4.2 Planning Calculus Type System

In this section we show how to use existing process calculi type systems for static analysis of planning problems. Type systems for processes calculi are used to prove various properties of processes. A type system is usually handcrafted for a specific calculus and thus we can not use an arbitrary type system for our planning process calculus. However, there is a generic process calculi type system scheme POLY$\star$ (Makholm and Wells, 2005) which works for many calculi including ours. Furthermore, POLY$\star$ has already been successfully used for static analysis (Jakubův and Wells, 2010).

A detailed description of POLY$\star$ is beyond the scope of this paper and thus we provide only a necessary background. POLY$\star$ provides a syntax to describe rewriting rule axioms. Here we just state that rewriting rules from the previous section can be easily described in this syntax. Once rewriting rules are given, POLY$\star$ automatically derives syntax of types together with a type system for the given calculus. Furthermore, an effective type inference algorithm is provided to compute a principal (most general) type of an arbitrary process.

For our purposes, it is enough to state that a POLY$\star$ type τ for our planning process calculus can be understood as a set of facts. POLY$\star$ defines a *typing relation*, written $\vdash Q : \tau$, which states that Q has type τ. The most important property of the typing relation is *subject reduction* which ensures that types are preserved under rewriting, that is, when $\vdash Q_0 : \tau$ and $Q_0 \to Q_1$ then $\vdash Q_1 : \tau$. Another property that interests us is that whenever $\vdash Q : \tau$ then τ contains all

[4] Briefly, for a delete effect $p \notin \mathsf{pre}(a)$, we can introduce a new fact p^- complementary to p and then provide two rewriting rule axioms for a; the first applies in states where p holds while the second in states where p^- holds.

the facts mentioned in Q. In particular, $\vdash \lceil s \rceil : \tau$ implies $s \subseteq \tau$ for any state s.

The previous paragraph suggests the following procedure. Given a STRIPS problem $\Pi = \langle P, A, I, G \rangle$, we can use POLY✶ to compute the principal type τ of the initial state process $\lceil I \rceil$. Hence $\vdash \lceil I \rceil : \tau$. Because types are preserved under rewriting, we know that whenever $\lceil I \rceil$ rewrites using an arbitrary many applications of $\rightarrow$ to some process Q, then also $\vdash Q : \tau$. In other words, whatever state s is reachable from I, we know that $s \subseteq \tau$. Hence when $G \not\subseteq \tau$ then Π is not solvable. The opposite implication, which would also allow us to recognize solvable problems, does not generally hold because POLY✶ type τ only over-approximates all reachable states.

In this way we can use POLY✶ types to recognize some unsolvable problems. We instantiate POLY✶ by translating actions to rewriting rules, and we use POLY✶ to compute the principal type τ of $\lceil I \rceil$. When $G \not\subseteq \tau$ then the problem is clearly unsolvable. Otherwise we can not conclude anything. This is the price we pay for effectiveness as POLY✶ types can be computed in polynomial time while planning is PSPACE-complete.

Our experiments have shown that POLY✶ type analysis is essentially equivalent to computing a planning graph with the delete effect relaxation. The POLY✶ type contains exactly the same facts as the last layer of a relaxed planning graph. A possible extensions of POLY✶ analysis that would give more precise results than relaxed planning graphs are left for future research. One of the advantages of using POLY✶ over relaxed planning graphs is in that we can rely on its already proved formal properties (subject reduction, principal typings) and that there is no need to implement equivalent methods because an effective type inference algorithm is already implemented.

5 PSM WITH PLAN ANALYSIS

This section describes improvements from the basic version of Algorithm 1 denoted in ***bold italics*** in the algorithm. We introduce two different methods (PSM+ and PSM✶) to incorporate the static analysis from the previous section into our PSM-based planner (Algorithm 1). The static analysis is used to analyze plans at the second line of the loop in the algorithm. Section 5.1 describes an encoding of an α-extensibility check into a planning problem, which is a technique shared by both the methods. It basically restates our previous work (Tožička et al., 2014a) required for the understanding of the next sections. Sections 5.2 and 5.3 describe in turn the methods PSM+ and PSM✶.

5.1 Plan Extensibility as Planning

First we describe how the static analysis from the previous section can be used to approximate α-extensibility of public plan σ. This is done by running the static analysis on a classical planning problem Π_σ constructed as follows. Problem Π_σ is similar to $\Pi \triangleright \alpha$ but it contains only a subset of its actions. Concretely, Π_σ contains all the internal actions of α but only those public or external actions which are mentioned in σ. Furthermore, actions from σ are interlinked using additional facts to ensure they are executed in the order suggested by σ. Formally, let a_i be the action from $A \triangleright \alpha$ which corresponds to the i-th action in σ. Let $mark_0, \ldots, mark_n$ be additional facts (n is the length of σ). Then Π_σ contains the following action b_i.

$$b_i = \langle\, \mathsf{pre}(a_i) \cup \{mark_{i-1}\},\ \mathsf{add}(a_i) \cup \{mark_i\},\ \mathsf{del}(a_i) \cup \{mark_{i-1}\} \,\rangle$$

Finally $mark_0$ is added to the initial state of Π_σ and $mark_n$ is added to the goal state. When action b_i is used, the mark is increased which enables action b_{i+1}. In the end we want all b_i's to be used, possibly interleaved with some α-internal actions which are just added to Π_σ without any additional changes. It can be proved that Π_σ is solvable iff σ is α-extensible (Tožička et al., 2014a).

5.2 Simplified Plan Analysis (PSM+)

The previous section suggests the following plan verification procedure. When agent α generates a new plan π_α, it sends its public projection $\pi^\star_\alpha$ to all the other agents. Once other agent β receives $\pi^\star_\alpha$, it uses the static analysis to check whether $\pi^\star_\alpha$ is β-extensible. That is, agent β constructs Π_σ (for $\sigma = \pi^\star_\alpha$) and executes the POLY✶ analysis yielding the type τ. When $G \not\subseteq \tau$ then agent β informs α that $\pi^\star_\alpha$ is not β-extensible. Otherwise agent β returns an unknown status back to agent α.

Once the initiator agent α receives back results from all the other agents it simply checks whether $\pi^\star_\alpha$ was *rejected* by, that is, found not β-extensible for, at least one agent. When the plan is rejected by at least one agent, then agent α simply drops the plan and directly continues with the next loop iteration by generating another plan. Optionally, agent α can incorporate plans generated by other agents as landmarks (last operation in the loop) provided these plans were not rejected.

5.3 Partial Plan Reuse (PSM✶)

Experiments with PSM+ showed that plan analysis had increased the number of solved problems when compared to the basic variant PSM without any plan analysis. However, there were some problems solved by PSM which were no longer solved by PSM+. A more detailed analysis revealed that in some cases a useful landmark was created from the beginning of a plan that was rejected in PSM+. This is because it can happen that a rejected plan is correct up to some point. The method introduced in this section tries to find a usable plan prefix and use it as a landmark.

The procedure starts as in the previous section, that is, agent α generates a new plan π_α and sends its public projection $\pi^\star_\alpha$ to all the other agents. Once Π_σ is constructed by other agent β, the POLY✶ analysis of Π_σ is executed yielding the type τ. When the last mark $mark_n$ is not in τ, not only we know that σ is surely not β-extensible, but the maximum mark present in τ gives us other insight into σ. Hence type τ is examined and the maximum i such that $mark_i \in \tau$ is found. From that we can conclude that it is not possible for β to follow the public plan σ to a state where the $(i+1)$-th action of σ can be executed. Hence this maximum i is returned as a result of β's analysis of σ back to agent α.

Finally, agent α collects analysis results from all the other agents and computes their minimum j. Plan σ is then stripped so that only the first j actions remain in it. This stripped plan is then sent to the other agents to be used as a landmark and to guide future plan search.

6 EXPERIMENTAL RESULTS

We have performed a set of experiments to evaluate an impact of plan verification on a PSM-based planner and also to compare our planners with another state-of-the-art multiagent planner[5]. We have decided to compare our approach with FMAP (Torreño et al., 2014) which uses well defined problems taken from International Planning Competition (IPC) problems. FMAP classifies facts as public or internal using a manual selection of public predicate names. In practice, FMAP public facts are a superset of MA-STRIPS public facts and thus FMAP classification is compatible with our algorithms. 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 minutes for each problem. The binary and source codes of our PSM-based planner are available on demand. Please contact us by email in order to obtain them.

Table 1: Number of problems solved by the compared planners.

Domain	FMAP	PSM	PSM+	PSM✶
Blocksworld (34)	19	**27**	26	26
Driverlog (20)	**15**	10	9	14
Elevators (30)	**30**	1	3	4
Logistics (20)	**10**	0	0	0
Openstacks (30)	23	**30**	**30**	**30**
Rovers (20)	**19**	7	14	14
Satellite (20)	**16**	6	13	9
Woodworking (30)	22	**27**	**27**	**27**
Zenotravel (20)	**18**	17	17	17
Total (224)	**172**	125	139	141

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. Nevertheless PSM+ performed better in two tightly-coupled domains as it was able to solve all the *Openstacks* problems and two additional *Woodworking* problems over FMAP. We can see that PSM✶ outperforms PSM+ in *Driverlog* and *Elevators* domains but it loses in *Satellite* domain. Moreover, there were eight other differences in individual problems, where half of them in each domain were in favor of each method and thus these differences are not reflected in the table. The results show that a possible enhancement of POLY✶ verification could bring even higher coverage. This will be part of our future research. Also note that PSM✶ is strictly better in coverage[7] than the basic variant PSM.

For a more detailed analysis of PSM variants we have chosen three domains with the highest coverage. Table 2 shows an average number of iterations and run times needed to find a solution for problems which were solved by all the three variants. Sequential times show how long it would take if all the agents share a single CPU, while the parallel time correspond to a

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

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

[7]However, there were still two individual problems which were solved by PSM but not by PSM✶. As this happens relatively rarely, it is left for future research to find out whether this is because of the approximation in POLY✶ analysis or whether it can happen that in some cases a useful landmark is constructed from a rejected plan part.

Table 2: Performance cost of PSM extended with plan verification. The table shows average values for all the solved problems. Run times are in seconds.

	PSM	PSM+	PSM✶
Driverlog			
Iterations	3.4	2	2.3
Sequential time	23.5	9.6	12.1
Parallel time	9.7	4.5	5.6
Openstacks			
Iterations	1	1	1
Sequential time	2.6	7.1	7.1
Parallel time	1.3	4.4	4.4
Woodworking			
Iterations	1.4	1.3	1.3
Sequential time	79.1	84.1	84.0
Parallel time	20.3	21.7	21.6

situation where each agent is equipped with its own CPU. The results show that in *Driverlog* domain the number of iterations decreased which also caused a decrease in run times. PSM+ achieved the best results for this domain. All problems of *Openstacks* domain have been solved during the first iteration even by the simplest version PSM. Therefore run times needed by the other versions are higher because of the time needed for the verification. Only a tiny decrease of iteration count in *Woodworking* domain could not outweigh the price for verification and thus the versions with plan verification are a bit slower than PSM. A slight increase of run time in PSM✶ over PSM+ is caused by additional landmarks which come from plans which were completely rejected by PSM+.

Table 3 compares run times needed to solve selected tasks solvable by all the planners. We can see that PSM-variants are able to find solution faster than FMAP in the case of complex problems.

Graph in Figure 2 compares times needed for planning and plan verification in PSM✶. It shows that the time needed for verification in PSM✶ is much smaller than the time needed for agent internal planning. The graph is constructed as follows. The x-axis in the graph shows total time needed to solve a problem, that is, planning together with verification. For each problem, planning and verification times are depicted as two values in the same column whose x-coordinate correspond to the total sequential time. Thus the sum of the two values in each column is always equal to the x-coordinate of the column.

Graphs in Figure 3 show, for each domain, an average number of iterations and an average amount of communication among all the agents measured in actions. The average values are computed only for the problems solved by all PSM, PSM+ and PSM✶ variants. The communication is measured as the number of actions communicated between each pair of agents. These actions are communicated during the verification and during the exchange of created public PSMs. During communication, each action is represented by its unique id and thus the number of actions communicated directly corresponds to the number of bytes.

Table 3: Comparison of run times on selected problems solved by all the planners. Times are in seconds, PSM times correspond to parallel times, and PSM variants have number of iterations in parenthesis.

	FMAP	PSM	PSM+	PSM✶
Driverlog				
p-01	**0.6**	2.2 (2)	2.3 (2)	2.3 (2)
p-05	**1.8**	34.1 (9)	4.0 (2)	6.5 (3)
p-08	11.9	11.6 (3)	**5.4** (2)	6.2 (2)
p-10	2.1	**3.0** (2)	4.2 (2)	4.6 (2)
p-13	16.2	14.3 (3)	**8.7** (2)	14.8 (3)
Openstacks				
p-01	1.4	**1.2** (1)	1.3 (1)	1.3 (1)
p-06	9.7	**1.1** (1)	1.7 (1)	1.7 (1)
p-11	51.0	**1.2** (1)	2.3 (1)	2.3 (1)
p-16	171.0	**1.2** (1)	4.3 (1)	4.5 (1)
p-21	497.0	**1.4** (1)	6.7 (1)	6.5 (1)
Woodworking				
p-01	**2.7**	4.7 (2)	5.6 (2)	5.6 (2)
p-06	200.3	**30.1** (2)	33.7 (2)	33.8 (2)
p-11	1.9	**1.8** (1)	2.3 (1)	2.3 (1)
p-21	**0.4**	1.3 (1)	1.5 (1)	1.5 (1)

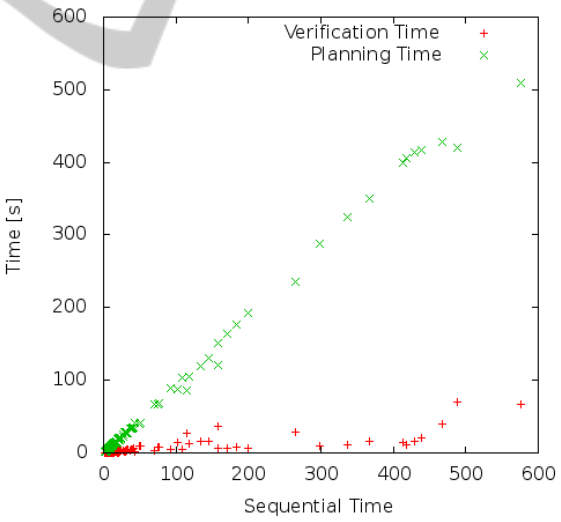


Figure 2: Verification and planning times in PSM✶.

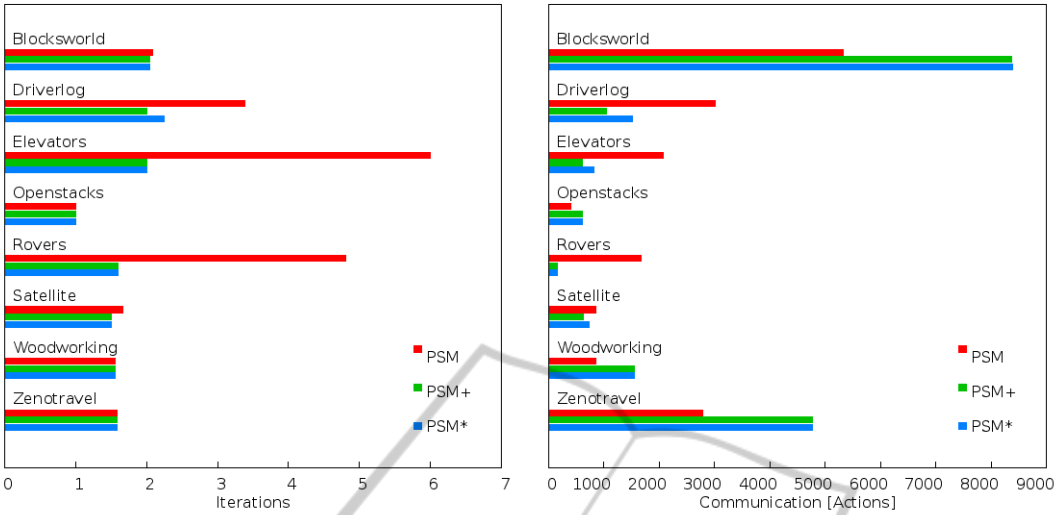


Figure 3: An average number of iterations (left) and an average amount of communication (right) for each domain in PSM, PSM+, and PSM✶ variants.

We can see that in some domains (e.g. *Blocksworld*) the communication grows substantially in PSM✶ and PSM+, while in other domains the communication is decreased. The reason for that is that in the first case all the plans are accepted and thus the verification process brings no advantage. In the domains where the verification helped, the overall communication in PSM+ and PSM✶ variants is smaller than in PSM. The domains where the verification was useful can be identified either from the overall results (Table 1) or from the average number of iterations (Figure 3, left) because the average number of iterations is decreased by success of the verification process (see domains *Driverlog*, *Elevators*, *Rovers*). In the domains where all the solved problems were solved in first few iterations even by the simplest PSM variant (*Blocksworld*, *Openstacks*, *Woodworking*, *Zenotravel*), the verification can not really help to decrease the number of iterations (as it is already small) and thus the verification only creates communication overhead. An exception is the *Satellite* domain where individual iteration numbers have a higher variance (which is not apparent from the average values in the graph).

Graph in Figure 4 shows amount of communication among the agents in a single selected problem (*Driverlog05*). This problem was chosen because it was solved by all the approaches but not in a trivial manner (in the first or second iteration). Each curve ends in the column that corresponds to the last iteration. We can see that the verification creates communication overhead in individual iterations but the total communication is smaller with verification because the number of iterations is decreased.

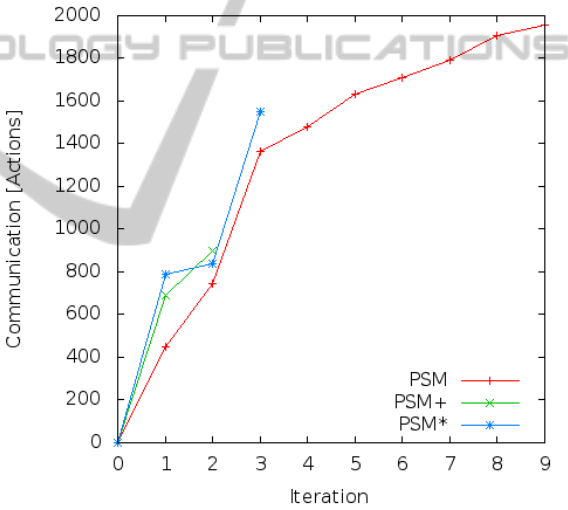


Figure 4: Amount of communication during solving of *Driverlog05*.

7 CONCLUSIONS

We have shown how integration of a static analysis based on process calculi type systems in validation phase of a planner based on merging of Planning State Machines strictly improves coverage of solved planning problem instances. Although the approach loses against a state-of-the-art multiagent planner, the results are promising. Moreover usage of the static analysis can improve other multiagent planning approaches using cooperation by coordination of partial agents' plans.

Furthermore, we have also extended our approach

with a new heuristic with notable improvements. This new heuristic will be part of our future research. In future research, we want also to focus on more precise static analysis by POLY✶ and therefore hypothetically less approximate test of the extensibility of partial plans.

ACKNOWLEDGEMENTS

This research was supported by the Czech Science Foundation (grant no. 13-22125S).

REFERENCES

Bhattacharya, S., Kumar, V., and Likhachev, M. (2010). Search-based path planning with homotopy class constraints. In Felner, A. and Sturtevant, N. R., editors, *SOCS*. AAAI Press.

Brafman, R. and Domshlak, C. (2008). From One to Many: Planning for Loosely Coupled Multi-Agent Systems. In *Proceedings of ICAPS'08*, volume 8, pages 28–35.

Fikes, R. and Nilsson, N. (1971). STRIPS: A new approach to the application of theorem proving to problem solving. In *Proc. of the 2nd International Joint Conference on Artificial Intelligence*, pages 608–620.

Hopcroft, J. E., Motwani, R., and Ullman, J. D. (2006). *Introduction to Automata Theory, Languages, and Computation (3rd Edition)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Jakubův, J. and Wells, J. B. (2010). Expressiveness of generic process shape types. In *TGC'10*, volume 6084 of *LNCS*, pages 103–119. Springer.

Makholm, H. and Wells, J. B. (2005). Instant polymorphic type systems for mobile process calculi: Just add reduction rules and close. In *ESOP'05*, volume 3444 of *LNCS*, pages 389–407. Springer.

Nissim, R. and Brafman, R. I. (2012). Multi-agent A* for parallel and distributed systems. In *Proc. of AAMAS'12*, AAMAS '12, pages 1265–1266, Richland, SC.

Štolba, M. and Komenda, A. (2014). Relaxation heuristics for multiagent planning. In *Proceedings of ICAPS'14*.

Torreño, A., Onaindia, E., and Sapena, s. (2014). Fmap: Distributed cooperative multi-agent planning. *Applied Intelligence*, 41(2):606–626.

Tožička, J., Jakubův, J., Durkota, K., Komenda, A., and Pěchouček, M. (2014a). Multiagent Planning Supported by Plan Diversity Metrics and Landmark Actions. In *Proceedings ICAART'14*.

Tožička, J., Jakubův, J., and Komenda, A. (2014b). Generating Multi-Agent Plans by Distributed Intersection of Finite State Machines. In *Proceedings ECAI'14*, pages 1111–1112.