TPLP **20** (5): 593–608, 2020. © The Author(s), 2020. Published by Cambridge University Press
doi:10.1017/S1471068420000289

Modelling Multi-Agent Epistemic Planning in ASP

ALESSANDRO BURIGANA, FRANCESCO FABIANO, AGOSTINO DOVIER
University of Udine, Udine, Italy
(*e-mail:* burigana.alessandro@spes.uniud.it)
(*e-mail:* {francesco.fabiano,agostino.dovier}@uniud.it)

ENRICO PONTELLI
New Mexico State University, Las Cruces, NM, USA
(*e-mail:* epontell@cs.nmsu.edu)

submitted 7 August 2020; accepted 10 August 2020

Abstract

Designing agents that reason and act upon the world has always been one of the main objectives of the Artificial Intelligence community. While for planning in "simple" domains the agents can solely rely on facts about the world, in several contexts, *e.g.*, economy, security, justice and politics, the mere knowledge of the world could be insufficient to reach a desired goal. In these scenarios, *epistemic* reasoning, *i.e.*, reasoning about agents' beliefs about themselves and about other agents' beliefs, is essential to design winning strategies. This paper addresses the problem of reasoning in multi-agent epistemic settings exploiting declarative programming techniques. In particular, the paper presents an actual implementation of a multi-shot *Answer Set Programming*-based planner that can reason in multi-agent epistemic settings, called PLATO (e**P**istemic mu**L**ti-agent **A**nswer se**T** programming s**O**lver). The ASP paradigm enables a concise and elegant design of the planner, w.r.t. other imperative implementations, facilitating the development of formal verification of correctness. The paper shows how the planner, exploiting an ad-hoc epistemic state representation and the efficiency of ASP solvers, has competitive performance results on benchmarks collected from the literature.

KEYWORDS: Epistemic Reasoning, Multi-Shot ASP, Planning, Multi-Agent, Possibilities

1 Introduction

The research area of *Reasoning about Actions and Change (RAC)* has been particularly active in recent years, motivated by the wider introduction of autonomous systems and the use of multi-agent techniques in a variety of domains (e.g., cyber-physical systems). The role of logic programming has been central to RAC research, especially thanks to the use of logic programming to formalize the semantics of high level action languages and to experiment with different extensions of such languages (Gelfond and Lifschitz 1993).

Over the years, several action languages (*e.g.*, $\mathcal{A}$, $\mathcal{B}$, and $\mathcal{C}$) have been developed, as discussed by Gelfond et al. (Gelfond and Lifschitz 1998). Each of these languages addresses important problems in RAC. Action languages have also provided the foundations for several successful approaches to automated planning (Castellini et al. 2001; Son et al. 2005).

In the special case of *multi-agent domains*, an agent's action may not just change

https://doi.org/10.1017/S1471068420000289 Published online by Cambridge University Press
CrossMark

the world (and possibly the agent's own knowledge), but also may change other agents' knowledge and beliefs. Similarly, the goals of an agent in a multi-agent domain, may involve not only reaching a desirable configuration of the world, but may also involve affecting the knowledge and beliefs of other agents about the world. While the literature about planning in multi-agent domains is rich (Durfee 1999; de Weerdt et al. 2003; Goldman and Zilberstein 2004; de Weerdt and Clement 2009; Dovier et al. 2013), relatively fewer efforts have explored the challenges of planning in multi-agent domains in presence of goals and actions that rely on and manipulate agents' knowledge and beliefs. In previous work (Baral et al. 2015), we proposed a high level action language, $m\mathcal{A}^*$, providing such features as: (i) actions that can change the world; (ii) actions that can impact either the knowledge of the agent or the beliefs and knowledge of other agents; (iii) actions that can affect agents' awareness of other events' occurrence. $m\mathcal{A}^*$ has received an updated semantics, based on *possibilities* (Fabiano et al. 2019; Fabiano et al. 2020), which offered several advantages, in terms of simplicity and compactness of state representations. Two different planners have been proposed by Le et al. and Fabinano et al. (Le et al. 2018; Fabiano et al. 2020), demonstrating the feasibility of planning in the domains described by $m\mathcal{A}^*$.

In this paper, as Baral et al. (Baral et al. 2010), we explore the use of logic programming, in the form of Answer Set Programming (ASP), to provide a novel implementation of a multi-agent epistemic planner. The implementation supports planning domains described using our possibilities-based multi-agent action language. The interest in this research direction derives from the desire of having a planner which is usable, efficient, and yet encoded using a declarative language. The declarative encoding allows us to provide formal proofs of correctness, which are presented in this paper. The declarative encoding will furnish a framework to explore a diversity of aspects of multi-agent epistemic planning, such as the impact of different optimizations (e.g., heuristics, avoidance of repeated states), the use of different semantics, and the introduction of extensions of the original action language.

The implementation relies on the multi-shot capabilities of the *clingo* solver. The encodings, a discussion about the nature of possibilities (Definition 2) and complete proofs of Propositions 1–3 are available at `http://clp.dimi.uniud.it/sw/`.

2 Multi-Agent Epistemic Planning

Let us begin by introducing the core elements of *Multi-agents Epistemic Planning (MEP)*. Let $\mathcal{AG}$ be a finite set of agents and $\mathcal{F}$ be a finite set of propositional variables, called *fluents*. Fluents allow us to describe the properties of the world in which the agents operate. A *possible world* is a representation of a possible configuration of the world, and it is described by a subset of $\mathcal{F}$ (intuitively, those fluents that are *true* in that world). Agents often have incomplete knowledge of the world, thus requiring the agent to deal with a set of possible worlds; the incomplete knowledge applies also to each agent's knowledge/beliefs about other agent's knowledge/beliefs. In epistemic planning, each action can be performed by an agent $\mathtt{ag} \in \mathcal{AG}$. The effect of an action can either change the physical state of the world (*i.e.*, the fluents) or the agents' beliefs. Specifically, we want to deal with the agents' beliefs about both the world and the other agents' beliefs.

https://doi.org/10.1017/S1471068420000289 Published online by Cambridge University Press

To this end we make use of a logic that is concerned with information change, namely *Dynamic Epistemic Logic (DEL)* (Van Ditmarsch et al. 2007).

We first introduce its syntax. A *fluent formula* is a propositional formula built using the fluents in $\mathcal{F}$. Fluent formulae allow us to express properties about a single possible world. To assert properties about what an agent believes, we use the modal operator $\mathbf{B}_{\mathsf{i}}$, where $\mathsf{i} \in \mathcal{AG}$. We read a formula $\mathbf{B}_{\mathsf{i}}\varphi$ as "agent i believes φ". Given a nonempty set of agents $\alpha \subseteq \mathcal{AG}$, the *group operators* $\mathbf{E}_\alpha$ and $\mathbf{C}_\alpha$, that intuitively represent the belief and the common belief of α, respectively, will be also used.

Definition 1 (*Belief formula*)
A *belief formula* is defined recursively as follows:

- A fluent formula is a belief formula.
- If φ_1, φ_2 are belief formulae, then $\neg\varphi_1$, $\varphi_1 \vee \varphi_2$ and $\varphi_1 \wedge \varphi_2$ are belief formulae.
- If φ is a belief formula and $\mathsf{ag} \in \mathcal{AG}$ is an agent, then $\mathbf{B}_{\mathsf{ag}}\varphi$ is a belief formula.
- If φ is a belief formula and $\emptyset \neq \alpha \subseteq \mathcal{AG}$, then $\mathbf{E}_\alpha\varphi$ and $\mathbf{C}_\alpha\varphi$ are belief formulae.

The semantics of DEL formulae is traditionally expressed using *pointed Kripke structures* (Kripke 1963); in previous work (Fabiano et al. 2019; Fabiano et al. 2020), we provided a semantics based on the concept of *possibilities*.

Definition 2 (*Possibility (Gerbrandy and Groeneveld 1997)*)
- A *possibility* u is a function that assigns to each fluent $\mathsf{f} \in \mathcal{F}$ a truth value $\mathsf{u}(\mathsf{f}) \in \{0, 1\}$ and to each agent $\mathsf{ag} \in \mathcal{AG}$ an information state $\mathsf{u}(\mathsf{ag}) = \sigma$;
- An *information state* σ is a set of possibilities.

Possibilities allow us to capture the concept of *epistemic state* (briefly, *e-state*). E-states consist of two components: information about the possible worlds and information about the agents' beliefs. Let u be a possibility. The assignment of truth values $\mathsf{u}(\mathsf{f})$ for each fluent $\mathsf{f} \in \mathcal{F}$ encodes a possible world; the assignment of information states to an agent $\mathsf{ag} \in \mathcal{AG}$ captures the beliefs of ag. Information states encode the same information represented by the edges of a Kripke structure; that is, an information state $\mathsf{u}(\mathsf{ag})$ is comparable to the set of worlds reached by ag from the world u in a Kripke structure. If $\mathsf{u}(\mathsf{f}) = 0$, then, in the possible world represented by u, the fluent f is false. Similarly, if $\mathsf{u}(\mathsf{ag}) = \{\mathsf{v}\}$, then (in the possibility u) the agent ag believes only the possibility v. Since possibilities are non-well-founded objects (*i.e.*, we do *not* require the sets to be well-founded), the concepts of *state* and *possible world* collapse. In fact, a possibility contains both the information of a possible world (the interpretation of the fluents) and the information about the agents' beliefs (represented by other possibilities). Hence, we denote the state/possible world that represents the *real world* as the *pointed possibility*. Due to space constraints, we refer the interested reader to the supplementary documents (available at `http://clp.dimi.uniud.it/sw/`) and to Gerbrandy et al. and Fabiano et al. (Gerbrandy and Groeneveld 1997; Fabiano et al. 2019) for a more complete discussion on the nature of possibilities.

Definition 3 (*Entailment w.r.t. possibilities (Fabiano et al. 2019)*)
Let the belief formulae $\varphi, \varphi_1, \varphi_2$, a fluent f, an agent ag, a (non-empty) group of agents α, and a possibility u be given.

i. $\mathsf{u} \models \mathsf{f}$ if $\mathsf{u}(\mathsf{f}) = 1$;

https://doi.org/10.1017/S1471068420000289 Published online by Cambridge University Press

ii. $\mathsf{u} \models \neg\varphi$ if $\mathsf{u} \not\models \varphi$;
iii. $\mathsf{u} \models \varphi_1 \vee \varphi_2$ if $\mathsf{u} \models \varphi_1$ or $\mathsf{u} \models \varphi_2$;
iv. $\mathsf{u} \models \varphi_1 \wedge \varphi_2$ if $\mathsf{u} \models \varphi_1$ and $\mathsf{u} \models \varphi_2$;
v. $\mathsf{u} \models \mathbf{B}_{\mathsf{ag}}\varphi$ if for each $\mathsf{v} \in \mathsf{u}(\mathsf{ag})$ it holds that $\mathsf{v} \models \varphi$;
vi. $\mathsf{u} \models \mathbf{E}_{\alpha}\varphi$ if for all $\mathsf{ag} \in \alpha$ it holds that $\mathsf{u} \models \mathbf{B}_{\mathsf{ag}}\varphi$;
vii. $\mathsf{u} \models \mathbf{C}_{\alpha}\varphi$ if $\mathsf{u} \models \mathbf{E}^{k}_{\alpha}\varphi$ for every $k \geq 0$, where $\mathbf{E}^{0}_{\alpha}\varphi = \varphi$ and $\mathbf{E}^{k+1}_{\alpha}\varphi = \mathbf{E}_{\alpha}(\mathbf{E}^{k}_{\alpha}\varphi)$.

We say that an agent *believes* a belief formula φ w.r.t. a given possibility if all of the possibilities within its information state entail φ. Common belief requires all agents in α to believe φ, that *all the agents in* α *believe* φ and so on *ad infinitum*.

Definition 4 (*MEP domain*)
A *multi-agent epistemic planning domain* is a tuple $D = \langle \mathcal{F}, \mathcal{AG}, \mathcal{A}, \varphi_i, \varphi_g \rangle$, where:

i. $\mathcal{F}$ is the finite set of *fluents* of D;
ii. $\mathcal{AG}$ is the finite set of *agents* of D;
iii. $\mathcal{A}$ represents the set of *actions* of D;
iv. φ_i is the belief formula that describes the *initial conditions*; and
v. φ_g is the belief formula that describes the *goal conditions* that we want to achieve.

A domain contains the information needed to describe a planning problem in a multi-agent epistemic setting. Given a domain D we refer to its elements through the parenthesis operator; *e.g.*, the fluent set of D will be denoted by $D(\mathcal{F})$. An *action instance* $\mathsf{a}\langle\mathsf{ag}\rangle \in D(\mathcal{AI}) = D(\mathcal{A}) \times D(\mathcal{AG})$ identifies the execution of action a by agent ag. Let $D(\mathcal{S})$ be the set of states reachable from $D(\varphi_i)$ with a finite sequence of actions. The *transition function* $\Phi : D(\mathcal{AI}) \times D(\mathcal{S}) \to D(\mathcal{S}) \cup \{\emptyset\}$ allows us to formalize the semantics of action instances (the result is the empty set if the action instance is not executable).

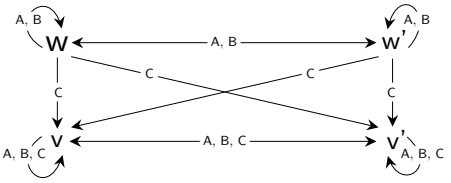
Possibilities are objects with a non-well-founded nature (Aczel 1988). This allows us to represent them by means of both a *picture* (*i.e.*, a *pointed graph*) and a *system of equations*, which are the standard representations for non-well-founded sets. In Figure 1 an example of a generic possibility illustrated using these two representations.

Definition 5 (*Decoration and picture (Aczel 1988)*)
A *decoration* of a graph $G = (V, E)$ is a function δ that assigns to each node $\mathsf{n} \in V$ a (non-well-founded) set δ_{n}, whose elements are the sets assigned to the successors of n in the graph, *i.e.*, $\delta_{\mathsf{n}} = \{\delta_{\mathsf{n}'} : (n, n') \in E\}$. Given a *pointed* graph (G, n) (*i.e.*, a graph with a node $\mathsf{n} \in V$ identified), if δ is a decoration of G, then (G, n) is a *picture* of the set δ_{n}.

Pictures also allow us to use graph terminology (edges, labels, reachability, etc.) when referring to possibilities. Given a possibility u and its picture $(G_{\mathsf{u}}, \mathsf{n}_{\mathsf{u}})$, when we refer to a "(labeled) edge" of u, we actually allude to its picture. When the context is clear we will use such terminology to refer directly to a possibility.

The non-well-founded nature of possibilities allow us to characterize the state equality through *bisimulation* (see Dovier (Dovier 2015) for a brief introduction). In fact, two decorations with bisimilar labeled graphs are represented by the same possibility.

https://doi.org/10.1017/S1471068420000289 Published online by Cambridge University Press



(a) Picture of w.

$$\begin{cases} \mathsf{w} & = \{(\mathsf{A}, \{\mathsf{w}, \mathsf{w}'\}), (\mathsf{B}, \{\mathsf{w}, \mathsf{w}'\}), (\mathsf{C}, \{\mathsf{v}, \mathsf{v}'\}), \mathsf{f}, \mathsf{g}, \mathsf{h}\} \\ \mathsf{w}' & = \{(\mathsf{A}, \{\mathsf{w}, \mathsf{w}'\}), (\mathsf{B}, \{\mathsf{w}, \mathsf{w}'\}), (\mathsf{C}, \{\mathsf{v}, \mathsf{v}'\}), \mathsf{g}, \mathsf{h}\} \\ \mathsf{v} & = \{(\mathsf{A}, \{\mathsf{v}, \mathsf{v}'\}), (\mathsf{B}, \{\mathsf{v}, \mathsf{v}'\}), (\mathsf{C}, \{\mathsf{v}, \mathsf{v}'\}), \mathsf{f}, \mathsf{h}\} \\ \mathsf{v}' & = \{(\mathsf{A}, \{\mathsf{v}, \mathsf{v}'\}), (\mathsf{B}, \{\mathsf{v}, \mathsf{v}'\}), (\mathsf{C}, \{\mathsf{v}, \mathsf{v}'\}), \mathsf{h}\} \end{cases}$$

(b) System of equations of w.

Fig. 1. Two equivalent representations of a generic possibility w. The possibility is expanded for clarity. Only "true" fluents are put in the set (rather than all pairs $(\mathsf{f}, 1), (\mathsf{g}, 1), (\mathsf{h}, 1)$). The interpretation of the fluents is the same in both figures.

Knowledge or belief. As pointed out in the previous paragraphs the modal operator $\mathbf{B}_{\mathsf{ag}}$ represents the worlds' relations in an e-state. As expected, different relations' properties imply different meanings for $\mathbf{B}_{\mathsf{ag}}$. In particular, we are interested in representing the agents' knowledge or beliefs. The accepted formalization for such concepts relies on the **S5** and **KD45** axioms, respectively. In fact, when a *relation*—represented by the *edges* in a Kripke structure and by the *information state* in a possibility—respects all the axioms presented in Table 1, it is called an **S5**-relation and it encodes the concept of knowledge; similarly, when the relation encodes all such axioms but **T**, we obtain a **KD45**-relation, that characterizes the concept of belief. Following these characterization we will refer to the logics of knowledge and belief as **S5** and **KD45** logic, respectively.

Property of B	Axiom
$\mathbf{B}_{\mathsf{ag}}\varphi \Rightarrow \varphi$	**T**
$\mathbf{B}_{\mathsf{ag}}\varphi \Rightarrow \mathbf{B}_{\mathsf{ag}}\mathbf{B}_{\mathsf{ag}}\varphi$	**4**
$\neg\mathbf{B}_{\mathsf{ag}}\varphi \Rightarrow \mathbf{B}_{\mathsf{ag}}\neg\mathbf{B}_{\mathsf{ag}}\varphi$	**5**
$\neg\mathbf{B}_{\mathsf{ag}}\bot$	**D**
$(\mathbf{B}_{\mathsf{ag}}\varphi \wedge \mathbf{B}_{\mathsf{ag}}(\varphi \Rightarrow \psi)) \Rightarrow \mathbf{B}_{\mathsf{ag}}\psi$	**K**

Table 1. *Knowledge and beliefs axioms.*

Intuitively the difference between the two logics is that an agent cannot *know* something that is not true in **S5**, but she can *believe* it in **KD45**. Our planner deals with e-states that comply with the axioms of **KD45**. However, it is possible to encode a domain in such a way that, when an action is performed, the resulting e-state is consistent with the axiom **T**. In this way we are able to reason within the **S5** logic. As explained in the following pages, we only require the initial state to satisfy all the **S5** axioms. As this introduction is not supposed to explore in depth this topic, we will not go into further detail and we address the interested reader to Fagin et al. (Fagin et al. 2004).

The language $m\mathcal{A}^{\rho}$. The planner EFP 2.0, introduced by Fabiano et al. in previous work (Fabiano et al. 2020), is able to reason on epistemic domains. Domain instances are encoded using the action language $m\mathcal{A}^{\rho}$ (Fabiano et al. 2019), in turn inspired by the language $m\mathcal{A}^{*}$ (Baral et al. 2015). The main difference between such languages lies

https://doi.org/10.1017/S1471068420000289 Published online by Cambridge University Press

in their semantics: while $m\mathcal{A}^*$ is based on Kripke structures and *updated models* (Baral et al. 2015), $m\mathcal{A}^\rho$ is given in terms of possibilities (Fabiano et al. 2020).

The languages $m\mathcal{A}^*$ and $m\mathcal{A}^\rho$ both allow three different types of action: i) *ontic* (or *world-altering*) actions that are used to change the properties of the world (*i.e.*, the truth value of fluents); ii) *sensing* and iii) *announcement* actions that are performed by an agent to change her beliefs about the world and to affect other agents' beliefs, respectively.

The action languages also allow to specify, for each action instance a, the *observability relation* of each agent. Namely, an agent may be *fully observant*, *partially observant*, or *oblivious* w.r.t. a. If an agent is fully observant, then she is aware of both the execution of the action instance and its effects; she is partially observant if she is only aware of the action execution but not of the outcomes; she is oblivious if she is ignorant of the execution of the action.

Given a domain D, an action instance $\mathsf{a} \in D(\mathcal{AI})$, a fluent literal f, a fluent formula ϕ and the belief formula φ, we introduce the syntax of $m\mathcal{A}^\rho$ as follows:

- **executable** a **if** φ: captures the *executability conditions*;
- a **causes** f **if** φ: captures the effects of *ontic* actions;
- a **determines** f **if** φ: captures the effects of *sensing* actions;
- a **announces** ϕ **if** φ: captures the effects of *announcement* actions;
- ag **observes** a **if** φ: captures *fully observant* agents for an action; and
- ag **aware_of** a **if** φ: captures *partially observant* agents for a given action.

Notice that if we do not state otherwise, an agent will be considered oblivious. Finally, statements of the form **initially** φ and **goal** φ capture the initial and goal conditions, respectively. The formulae φ_i and φ_g of a domain are obtained by a conjunction of the initial conditions and of the goal conditions, respectively.

Finitary S5 theories. Given a generic belief formula φ it is possible to generate infinitely many (initial) states that satisfy φ (see Son et al. (Son et al. 2014) for a complete introduction). To overcome this problem, we use the following notion and result.

Definition 6 (*Finitary **S5**-theory (Son et al. 2014)*)
Let ϕ be a fluent formula and let $\mathsf{i} \in \mathcal{AG}$ be an agent. A *finitary **S5**-theory* is a collection of formulae of the form (we use the short form $\mathbf{C}\ \phi$ instead of $\mathbf{C}_{\mathcal{AG}}\phi$):

$$(i)\ \phi \qquad (ii)\ \mathbf{C}\ \phi \qquad (iii)\ \mathbf{C}\ (\mathbf{B}_\mathsf{i}\phi \vee \mathbf{B}_\mathsf{i}\neg\phi) \qquad (iv)\ \mathbf{C}\ (\neg\mathbf{B}_\mathsf{i}\phi \wedge \neg\mathbf{B}_\mathsf{i}\neg\phi)$$

Moreover, we require each fluent $\mathsf{f} \in \mathcal{F}$ to appear in at least one of the formulae (ii)–(iv) (for at least one agent $\mathsf{i} \in \mathcal{AG}$).

As shown by Son et al., a finitary **S5**-theory has finitely many S5-models up to equivalence (*i.e.*, bisimulation). We therefore require that the set of formulae $\{\varphi \mid [\textbf{initially}\ \varphi] \in D\}$ is a *finitary **S5**-theory*. Moreover, in Section 4 we explain how the generation of a unique initial state is achieved. It is important to notice that this requirement applies only when the initial state description is given by means of a set of belief formulae. On the other hand, whenever the initial state is explicitly described, we do not impose any limitation. This allows us to simplify the initial state generation w.r.t. some other approaches (Van Der Hoek and Wooldridge 2002; Bolander and Andersen 2011; Löwe et al. 2011), where the initial e-state is assumed to be explicitly described.

https://doi.org/10.1017/S1471068420000289 Published online by Cambridge University Press

3 Multi-Shot Solving in ASP

A general program P in the language ASP is a set of rules r of the form:

$$a_0 \leftarrow a_1, \ldots, a_m, not\ a_{m+1}, \ldots, not\ a_n$$

where $0 \leq m \leq n$ and each element a_i, with $0 \leq i \leq n$, is an *atom* of the form $p(t_1, \ldots, t_k)$, p is a predicate symbol of arity k and $t_1, \ldots, t_k$ are terms built using variables, constants and function symbols. Negation-as-failure (naf) literals are of the form $not\ a$, where a is an atom. Let r be a rule, we denote with $h(r) = a_0$ its *head*, and $B^+(r) = \{a_1, \ldots, a_m\}$ and $B^-(r) = \{a_{m+1}, \ldots, a_n\}$ the positive and negative parts of its *body*, respectively; we denote the body with $B(r) = \{a_1, \ldots, not\ a_n\}$. A rule is called a *fact* whenever $B(r) = \emptyset$; a rule is a *constraint* when its head is empty ($h(r) = \mathsf{false}$); if $m = n$ the rule is a *definite rule*. A *definite program* consists of only definite rules.

A term, atom, rule, or program is said to be *ground* if it does not contain variables. Given a program P, its *ground instance* is the set of all ground rules obtained by substituting all variables in each rule with ground terms. In what follows we assume atoms, rules and programs to be *grounded*. Let M be a set of ground atoms ($\mathsf{false} \notin M$) and let r be a rule: we say that $M \models r$ if $B^+(r) \not\subseteq M$ or $B^-(r) \cap M \neq \emptyset$ or $h(r) \in M$. M is a *model* of P if $M \models r$ for each $r \in P$. The *reduct* of a program P w.r.t. M, denoted by P^M, is the definite program obtained from P as follows: (i) for each $a \in M$, delete all the rules r such that $a \in B^-(r)$, and (ii) remove all naf-literals in the the remaining rules. A set of atoms M is an *answer set* (Gelfond and Lifschitz 1988) of a program P if M is the minimal model of P^M. A program P is *consistent* if it admits an answer set.

We will make use of the multi-shot declarations for ASP, i.e. statements of the form `#program` $sp(p_1, \ldots, p_k)$, where sp is the name of the sub-program and the p_i's are its parameters (Gebser et al. 2019). Precisely, if R is a *list* of non-ground rules and declarations, with $R(sp)$ we denote the sub-program consisting of all the rules following the statement up to the next program declaration (or the end of the list). If the list does not start with a declaration, the default declaration `#base` is implicitly added by *clingo*.

An ASP program R is *extensible* if it contains declarations of the form `#external` $a : B$, where a is an atom and B is a rule body. These declarations identify a set of atoms that are outside the scope of traditional ASP solving (e.g., they may not appear in the head of any rule). When we set a to true we can *activate* all the rules r such that $a \in B^+(r)$. Splitting the program allows us to control the grounding and solving phases of each sub-program by explicit instructions using a Python script.

4 Modeling Epistemic Multi-agent Planning using ASP

We present a multi-shot ASP encoding for solving a multi-agent epistemic planning domain $D = \langle \mathcal{F}, \mathcal{AG}, \mathcal{A}, \varphi_i, \varphi_g \rangle$ (Definition 4) upon the possibilities based semantics described by Fabiano et al. (Fabiano et al. 2020). Its core elements are: the entailment of DEL formulae, the generation of the initial state and the transition function. The encoding implements a breadth-first search exploiting the multi-shot capabilities of *clingo*.

Epistemic states. As we discussed in Section 2, the elements that we need to encode are the possible worlds and the agents' beliefs. We use atoms of the form `possible_world(T,`

https://doi.org/10.1017/S1471068420000289 Published online by Cambridge University Press

`R, P)` and `believes(T1, R1, P1, T2, R2, P2, AG)`, respectively. Intuitively, the first atom identifies a possibility with a triple $(\mathtt{T}, \mathtt{R}, \mathtt{P})$, while the second encodes an "edge" between the possibilities $(\mathtt{T1}, \mathtt{R1}, \mathtt{P1})$ and $(\mathtt{T2}, \mathtt{R2}, \mathtt{P2})$, labeled with the agent `AG`.

Let us now focus in more detail on `possible_world(T, R, P)`. `P` is the index of the possibility. The variables `T` and `R` represent the *time* and the *repetition* of the possibility `P`, respectively. It is important to notice that these two parameters are necessary to uniquely identify a possibility during the solving process. The first parameter tells us *when* `P` was created: a possibility with time `T` is created after the execution of an action at time `T`. At a given time, it could be the case that two (or more) possibilities share both the values of `T` and `P`. Thus, a third value, the repetition `R`, is introduced with the only purpose to disambiguate between these cases. The update of repetitions will be explained during the analysis of the transition function.

Intuitively, the index `P` is used during the generation of the initial state to name the initial possible worlds. Afterwards, when an action is performed, we create new possibilities by updating the values of `T` and `R`. We do not need to modify the value of `P` as well, since the update of time and repetition is designed to be univocal for each `P`.

Let `ag` be an agent and `u` and `v` be two possibilities represented by the triples $(\mathtt{Tu}, \mathtt{Ru}, \mathtt{Pu})$ and $(\mathtt{Tv}, \mathtt{Rv}, \mathtt{Pv})$, respectively. Then, we encode the fact that $\mathtt{v} \in \mathtt{u}(\mathtt{ag})$ with the atom `believes(Tu, Ru, Pu, Tv, Rv, Pv, ag)`.

The truth value of each fluent is captured by an atom of the form `holds(Tu, Ru, Pu, F)`. The truth of such atom captures the fact that $\mathtt{u}(\mathtt{F}) = 1$. Finally, we specify the pointed possibility, for a given time `T`, using atoms of the form `pointed(T, R, P)`. For readability purposes, in the following pages we will identify a possibility `u` by `Pu` rather than by the triple $(\mathtt{Tu}, \mathtt{Ru}, \mathtt{Pu})$, when this will cause no ambiguity.

Entailment. To verify if a given DEL formula `F` is entailed by a possibility, we use the predicate `entails(P, F)`, defined below (with some simplifications for readability).

```
entails      (P,           F)  :- holds(P,F), fluent(F).

entails      (P,      neg(F))  :- not entails(P,F).
entails      (P,  and(F1,F2))  :- entails(P,F1), entails(P,F2).
entails      (P,   or(F1,F2))  :- entails(P,F1).
entails      (P,   or(F1,F2))  :- entails(P,F2).

not_entails  (P1,    b(AG,F))  :- not entails(P2,F), believes(P1,P2,AG).
entails      (P,     b(AG,F))  :- not not_entails(P,b(AG,F)).

not_entails  (P1,    c(AGS,F))  :- not entails(P2,F), reaches(P1,P2,AGS).
entails      (P,     c(AGS,F))  :- not not_entails(P,c(AGS,F)).
```

The encoding makes use of an auxiliary predicate `not_entails` to check whether a given formula `F` is not entailed by a possibility `P1`. For formulae of the type `b(AG, F)` we require that all of the possibilities believed by `AG` entail `F`. Similarly, for formulae of the type `c(AGS, F)` (where `AGS` represents a set of agents) we require that all of the possibilities *reached* by `AGS` entail `F`. A possibility `P1` reaches `P2` if it satisfies the following rules (where `contains`/2 is defined by a set of facts):

```
reaches(P1,P2,AGS):-believes(P1,P2,AG),contains(AGS,AG).
reaches(P1,P2,AGS):-believes(P1,P3,AG),contains(AGS,AG),reaches(P3,P2,AGS).
```

https://doi.org/10.1017/S1471068420000289 Published online by Cambridge University Press

Initial state generation. The initial state is set at time 0. Since we require the initial state to be a model of a finitary **S5**-theory, we assume the initial conditions to be DEL formulae of the form (i)–(iv) (Definition 6). Let us analyze how such formulae shape the initial state. Let ψ be a fluent formula, let $\mathsf{f} \in D(\mathcal{F})$ be a fluent and let $\mathsf{i} \in D(\mathcal{AG})$ be an agent. Consider a $m\mathcal{A}^\rho$ statement of the form [**initially** φ] $\in D$; we have five cases:

1. $\varphi \equiv \mathsf{f}$ ($\neg\mathsf{f}$): f must (not) hold in the pointed possibility.
2. $\varphi \equiv \mathbf{C}\ \mathsf{f}$ ($\neg\mathsf{f}$): f must (not) hold in each possibility of the initial state.
3. $\varphi \equiv \mathbf{C}\ \psi$: if ψ is a fluent formula that is *not* a fluent literal, then it must be entailed from each possibility of the initial state.
4. $\varphi \equiv \mathbf{C}\ (\mathbf{B}_\mathsf{i}\psi \vee \mathbf{B}_\mathsf{i}\neg\psi)$: there can be no two possibilities u and v such that $\mathsf{v} \in \mathsf{u}(\mathsf{i})$ and ψ is entailed by only one of them. Intuitively, this type of formula expresses the fact that agent i knows whether ψ is true in the initial state.
5. $\varphi \equiv \mathbf{C}\ (\neg\mathbf{B}_\mathsf{i}\psi \wedge \neg\mathbf{B}_\mathsf{i}\neg\psi)$: this type of formula expresses the fact that agent i does *not* know whether ψ is true or false in the initial state. Hence, given a possibility u, there must exist $\mathsf{v} \in \mathsf{u}(\mathsf{i})$ such that $\mathsf{u} \models \psi$ and $\mathsf{v} \not\models \psi$ (or $\mathsf{u} \not\models \psi$ and $\mathsf{v} \models \psi$).

Formulae of types 1–3 are used to build the fluent sets of the possible worlds within the initial state. A fluent f is *initially known* if there exists a statement [**initially C** (f)] or [**initially C** ($\neg\mathsf{f}$)]. In the former case, all agents will know that f is true, whereas in the latter that f is false. If there are no such statements for f, then it is said to be *initially unknown*. Let *uk* be the number of initially unknown fluents: we consider 2^{uk} initial possible worlds, addressed by an integer index $\mathtt{P} \in \{1, \ldots, 2^{uk}\}$, one for each possible truth combination of such fluents. For each initial possibility `P` and each initially known fluent `F`, we create an atom `holds(0, 0, P, F)`, since it is common belief between all agents that `F` is true (we deal with negated fluents similarly). Moreover, through the atoms `holds` we generate all the possible truth combinations for initially unknown fluents and we assign each one of them to an initial possibility. We require all the combinations to be different, thus each initial possibility represents a unique possible world.

An initial possibility is said to be *good* if it entails all of the formulae of type 3. We create a possible world `possible_world(0, 0, P)` for every *good* initial possibility `P`. The initial pointed possibility is specified by `pointed(0, 0, PP)`, where `PP` is the (unique) *good* initial possibility that entails all of the type 1 formulae. Finally, formulae of type 4 are used to filter out the edges of the initial state. Let `P1` and `P2` be two *good* initial possibilities; the atom `believes(0, 0, P1, 0, 0, P2, AG)` holds if there are no initial type 4 formulae ψ such that `P1` and `P2` do *not* agree on ψ. The construction of the initial state is achieved by filtering out the edges of a complete graph—*i.e.*, being $\mathcal{G}$ the set of *good* initial possibilities, $\forall \mathsf{u} \in \mathcal{G}, \forall \mathsf{ag} \in \mathcal{AG}$ we have that $\mathsf{u}(\mathsf{ag}) = \mathcal{G}$. We can observe that type 5 formulae do not contribute to this filtering, hence we do not consider them in the initial state generation.

Transition function. The transition function calculates the resulting state after the execution of an action at time $\mathtt{T} > 0$. There are three types of actions; the implementation of executability conditions is the same for all of them. For example, suppose that at time `T` we execute the ontic action `act`: the statement [`act` **causes** f **if** φ] tells us that in order to apply the action effect f on a possibility `u` we first need to satisfy the condition $\mathsf{u} \models \varphi$. To this end we introduced the predicate `is_executable_effect(T, ACT, T2,`

https://doi.org/10.1017/S1471068420000289 Published online by Cambridge University Press

R2, P2, E). If such an atom holds, then it denotes that the effect E of the action ACT performed at time T is executable in the possibility (T2, R2, P2). Without loss of generality, we represent an action instance by a unique action (using fresh actions names). Let us describe how we have modeled these actions in ASP (following the semantics described by Fabiano et al.).

Ontic actions. Let ACT be an ontic action executed at time T and let u = (T-1, RP, PP) be the pointed possibility at time T-1. Intuitively, when an ontic action is executed, the resulting possibility u′ is calculated by applying the action effects on u and also on the possibilities w ∈ u(ag), for each fully observant agent ag; and so on, recursively. Hence, we apply the action effects to all of the possibilities w that are *reachable with a path labeled with only fully observant agents* (briefly denoted as *fully observant path*). This concept is key to understand how the possible worlds are computed. Then possible_world is defined as follows:

```
possible_world  (T, R2 + MR + 1, P2):-
                pointed(T-1, RP, PP), possible_world(T2, R2, P2), T2<T,
                reaches(T-1, RP, PP, T2, R2, P2, AGS), subset(AGS, F_ACT).
```

where MR is the maximum value of the parameter *repetition* among all the possibilities at time T-1 and F_{ACT} represents the set of fully observant agents of ACT. Hence, if (T2, R2, P2) is a possibility that is reachable by a fully observant path at time T-1, then we create a new possibility (T, R2 + MR + 1, P2). When the body of the rule is satisfied, we say that P2 is *updated*. For short we will refer to the updated version of P2 as P2′. The time corresponds to the step number when the possibility was created; the repetition is calculated by adding to R2 the maximum repetition found at time T-1, plus one; finally, P2 is the name of the new possibility.

The pointed possibility at time T is pointed(T, 2*MR+1, PP). Notice that, since the maximum repetition at time 0 is 0 (by construction of the initial state) and since at time T we set the repetition of the pointed possibility to 2*MR+1, it follows that the maximum repetition at each time is associated to the pointed possibility itself. In this way we are able to always create a unique triple of parameters for each new possibility. At the moment, the plans that PLATO can handle in reasonable times have lengths that limit the exponential growth of such value within an acceptable range. In fact, even for the largest instance that was tested on EFP 2.0 (Fabiano et al. 2020), the length of the optimal plan was less than 20 (PLATO could not find a solution for such instance before the timeout). Nonetheless, we plan a more efficient design of the update of the repetition values through hashing functions or bit maps that would limit the growth of the repetition to a polynomial rate. This would achieve a polynomial growth of the repetition value, allowing the solver to handle much longer plans.

Next, we must state which fluents hold in the new possibilities. For each fluent F that is an executable effect of ACT, we impose holds(P2′, F) (and similarly for negative effects). The remaining fluents will hold in the updated possibility only if they did in the old one.

Finally, we deal with the agents' beliefs. Let P1, P2 be two updated possibilities and let AG be a fully observant agent. If believes(P1, P2, AG) holds, we impose believes(P1′, P2′, AG). Otherwise, if AG is oblivious, we impose believes(P1′, P2, AG) exploiting the already calculated possibility P2 to reduce the number of possible_world atoms.

https://doi.org/10.1017/S1471068420000289 Published online by Cambridge University Press

Sensing/Announcement actions. The behaviour of sensing and announcement actions is similar (as shown by Fabiano et al. (Fabiano et al. 2020)). The generation of the possible worlds is also similar to that of ontic actions. Let `ACT` be a sensing or an announcement action and let `PP` and `P2` be two possibilities such that `PP` is the pointed one at time `T-1` and `P2` is reachable from `PP`. We update `P2` in the following cases:

1. `P2` = `PP` (here we also set `P2′` as the pointed possibility at time `T`);
2. `P2` is reached by a fully observant path and it is consistent with the effects of `ACT`;
3. `P2` is reached by a path that starts with an edge labeled with a partially observant agent and that does *not* contain oblivious agents.

The pointed possibility must always be updated, in order for it to be consistent with the change of the agents' beliefs after the action is performed (that is, we do not want to carry old information obtained in previous states). Similarly to ontic actions, condition 2 deals with the possibilities believed by fully observant agents; if `ag` is fully observant, then she must only believe those possible worlds that are consistent with the effects of `ACT`. Finally, condition 3 deals with partially observant agents: since such an agent is not aware of the action's effects, we do not impose `P2′` to be consistent with the action's effects. Also, we restrict the first edge to be labeled by a partially observant agent in order to avoid the generation of superfluous possible worlds (namely, worlds that are not believed by any agent). In fact, the contribution to the update of the possible worlds by means of fully observant agents is entirely captured by condition 2.

We create a possible world `P2′` at time `T` for each `P2` that satisfies one of the conditions above. Since sensing and announcement actions do not alter the physical properties of the world, we impose `holds(P2′, F)` if `holds(P2, F)`, for each fluent `F` (inertia).

Let `AG` be a partially observant agent. If `believes(P1, P2, AG)` holds, then we will impose `believes(P1′, P2′, AG)`, since partially observant agents are not aware of the effects of the action. If `AG` is fully observant, we also add the condition that `P1` and `P2` are both (or neither) consistent with the effects of the actions. The purpose of this condition is twofold: first, we update the beliefs of the fully observant agents; second, we maintain the beliefs of partially observant agents w.r.t. the beliefs of the fully observant ones. We deal with oblivious agents exactly as for ontic actions.

Optimizations. In order to minimize the amount of grounded `possible_world` atoms, we designed the function so as to reuse, whenever possible, an already computed possibility. In this way, we efficiently deal with the beliefs of oblivious agents.

We were also able to significantly speed up the initial state generation by imposing a complete order between the initial possible worlds w.r.t. their fluents. Specifically, let `P1` and `P2` be two initial possibilities. Let `MFi = #max { F : holds(Pi, F), not holds(Pj, F) }`, with `i` $\neq$ `j`. Then we impose that if `P1` $<$ `P2`, then it must also hold that `MF1` $<$ `MF2`. Since it could be the case that there exist finitely many initial states, by implementing this constraint we are able to generate a unique initial state while discarding the (possible) other equivalent ones.

Multi-shot encoding. Following the approach of Gebser et al. (Gebser et al. 2019) we divided our ASP program into three main sub-programs, where the parameter `t` stands for the execution time of the actions: (i) `base`: it contains the rules for the generation of

https://doi.org/10.1017/S1471068420000289 Published online by Cambridge University Press

the initial state (`t` $= 0$), alongside with the instance encoding; (ii) `step(t)`: it deals with the plan generation (`t` > 0) and with the application of the transition function; and (iii) `check(t)`: it verifies whether the goal is reached at time `t` ≥ 0.

The sub-program `check(t)` contains the external atom `query(t)` that is used in the constraint: `:- not entails(t, R, P, F), pointed(t, R, P), goal(F), query(t).` The atom `query(t)` allows the solver to activate the constraint above only at time `t` (with the method `assign_external`) and to deactivate it when we move to time `t` $+ 1$ (method `release_external`). Using the Python script provided by Gebser et al., we first ground and solve the sub-program `base` and we check if the goal is reached in the initial state (`t` $= 0$); in the following iterations, the sub-programs `step(t)` (`t` > 0) are grounded and solved; we check the goal constraint until the condition is satisfied.

5 Experimental Evaluation

In this Section we compare PLATO to the multi-agent epistemic planner EFP 2.0 presented in previous work (Fabiano et al. 2020). All the experiments were performed on a 3.60GHz Intel Core i7-4790 machine with 32 GB of memory and with Ubuntu 18.04.3 LTS, imposing a time out (`TO`) of 25 minutes and exploiting ASP's parallelism on multiple threads. All the results are given in seconds. From now on, to avoid unnecessary clutter, we will make use of the following notations:

- L: the length of the optimal plan;
- d: the upper bound to the depth of nested modal operators **B** in the DEL formulae;
- K-BIS/P-MAR: the solver EFP 2.0 using the best configuration based on Kripke structures and possibilities, respectively;
- `single`/`multi`: PLATO using the single-shot/multi-shot encoding, respectively;
- `many`/`frumpy`: `multi` using the *clingo*'s configuration *many*/*frumpy*, respectively;
- `bis`: `multi` implemented with a visited state check based on bisimulation (following the implementation of Dovier (Dovier 2015)).

We considered various domains collected from the literature (Kominis and Geffner 2015; Huang et al. 2017), such as *Collaboration and Communication* (**CC**), *Selective Communication* (**SC**), *Grapevine* (**Gr**), *Assembly Line* (**AL**), and *Coin in the Box* (**CB**). The full description of these domains is reported in the supplementary documents and it can also be found in previous work by Fabiano et al. (Fabiano et al. 2020).

We report only the results of the *clingo*'s search heuristic configurations *many* and *frumpy* as they were the most performing ones in our set of benchmarks. Although generally they show a similar behaviour, as shown in Table 2a, in larger instances the time results differ substantially. In the results, when only `multi` is specified, we considered the most efficient configuration on the specific domain.

To evaluate the behaviour of PLATO w.r.t. the entailment of DEL formulae, we exploited the **AL** domain (Table 2d), where the executability conditions of the actions have depth d. The entailment of belief formulae with higher depth is handled efficiently by PLATO, although the use of common believes in the executability conditions leads to worst results. This is due to the fact that the number of `reached` atoms is substantially higher than the number of `believes` atoms (required in the entailment of **C** and **B** formulae, respectively). Notice that in ASP the entailment of each formula, independently

https://doi.org/10.1017/S1471068420000289 Published online by Cambridge University Press

SC: $\|\mathcal{AG}\| = 9$, $\|\mathcal{F}\| = 12$, $\|\mathcal{A}\| = 14$				
L	`many`	`frumpy`	K-BIS	P-MAR
4	.24	.24	.03	**.007**
6	2.56	2.49	.16	**.04**
8	36.79	38.34	4.23	**.30**
9	204.52	146.343	5.79	**.83**
10	TO	839.27	7.36	**1.78**

(a) Runtimes for Selective Communication.

Gr: $\|\mathcal{AG}\| = 3$, $\|\mathcal{F}\| = 9$, $\|\mathcal{A}\| = 24$				
L	Total	Ground	Solve	Atoms
3	.97	.60	.36	28'615
4	4.25	2.24	2.01	42'022
5	32.83	2.52	30.31	71'482
6	211.69	5.27	206.41	140'305
7	1066.80	16.94	1049.86	302'623

(b) Runtimes for Grapevine.

CB: $\|\mathcal{AG}\| = 3$, $\|\mathcal{F}\| = 8$, $\|\mathcal{A}\| = 21$				
L	`multi`	`bis`	K-BIS	P-MAR
2	.11	.11	.006	**.001**
3	.20	.24	.10	**.02**
5	1.21	4.21	1.44	**.37**
6	6.69	31.82	14.62	**2.93**
7	46.48	278.80	38.26	**6.99**

(c) Runtimes for Coin in the Box.

AL: $\|\mathcal{AG}\| = 2$, $\|\mathcal{F}\| = 4$, $\|\mathcal{A}\| = 6$			
d	`multi`	K-BIS	P-MAR
2	14.89	.42	**.07**
4	15.63	.64	**.11**
6	15.96	13.51	**2.44**
8	**17.55**	883.87	150.92
C	128.02	.43	**.08**

(d) Runtimes for Assembly Line.

	CC_1: $\|\mathcal{AG}\| = 2$, $\|\mathcal{F}\| = 10$, $\|\mathcal{A}\| = 16$				**CC**_2: $\|\mathcal{AG}\| = 3$, $\|\mathcal{F}\| = 13$, $\|\mathcal{A}\| = 24$			
L	`single`	`multi`	K-BIS	P-MAR	`single`	`multi`	K-BIS	P-MAR
3	48.74	6.52	.08	**.02**	153.76	14.07	.13	**.03**
4	188.32	8.74	.16	**.03**	TO	28.02	.54	**.10**
5	TO	7.68	1.14	**.16**	TO	16.13	4.89	**.60**
6	1222.67	10.83	4.42	**0.64**	TO	14.84	12.66	**1.71**
7	TO	30.08	16.06	**2.61**	TO	56.48	142.06	**12.37**

(e) Runtimes for Collaboration and Communication.

Table 2. *(a) Comparison of* ***frumpy***, ***many*** *and EFP 2.0 on* ***SC****. (b) Total, grounding and solving times for* ***Gr*** *using* ***multi****. The last column reports the number of grounded atoms. (c) Comparison of* ***multi*** *and* ***bis*** *on* ***CB****. (d) Comparison of PLATO and EFP 2.0 on* ***AL*** *(****C*** *identifies that the executability conditions are expressed through common believes). (e) Comparison of* ***single***, ***multi*** *and EFP 2.0 on* ***CC****.*

from its depth, is handled by a grounded atom and, therefore, a higher depth does not impact the solving process. On the other hand, the entailment in EFP 2.0 is handled by exploring all the paths of length d of the state, causing higher cost performances during each entailment check.

To investigate the contribution of the grounding and solving phases, we summed the computation times of the *clingo* functions `ground()` and `solve()` for each iteration. Table 2b shows a major contribution of the solving phase, hence indicating an efficient grounding. This permitted us to consider larger instances and to compete with other imperative approaches. The implementation of bisimulation within the multi-shot encoding leads to less efficient results (as shown in Table 2c), due to a much heavier contribution of the grounding phase.

Finally, we compare the single-shot/multi-shot encodings in Table 2e. The latter approach leads to significantly better results: the *clingo*'s option `-stat` revealed a smaller number of conflicts in the majority of the benchmarks. As explained by Gebser et al. (Gebser et al. 2019), this is due to the reuse of *nogoods* learnt from previous solving steps.

https://doi.org/10.1017/S1471068420000289 Published online by Cambridge University Press

6 Correctness of PLATO w.r.t. $m\mathcal{A}^{\rho}$

Declarative languages such as ASP allow a high-level implementation, facilitating the derivation of a formal verification of correctness of the planner. Consider a domain D; we denote the set of the belief formulae that can be built using the fluents in $D(\mathcal{F})$ and the propositional/modal operators by $D(\mathcal{BF})$. We denote the transition function of PLATO by $\Gamma : D(\mathcal{AI}) \times D(\mathcal{S}) \to D(\mathcal{S}) \cup \{\emptyset\}$ (where $D(\mathcal{AI})$ and $D(\mathcal{S})$ are defined as in Section 2). Finally, we express the entailment w.r.t. $m\mathcal{A}^{\rho}$ and PLATO with $\models_{\Phi}$ and $\models_{\Gamma}$, respectively. Each main component of the planner is addressed by a relative Proposition.

Proposition 1 (*PLATO entailment correctness w.r.t.* $m\mathcal{A}^{\rho}$)
Given a possibility $\mathsf{u} \in D(\mathcal{S})$ we have that $\forall \psi \in D(\mathcal{BF})$ $\mathsf{u} \models_{\Phi} \psi$ iff $\mathsf{u} \models_{\Gamma} \psi$.

Proposition 2 (*PLATO initial state construction correctness w.r.t.* $m\mathcal{A}^{\rho}$)
Given two possibilities $\mathsf{u}, \mathsf{v} \in D(\mathcal{S})$ such that u is the initial state in $m\mathcal{A}^{\rho}$ and v is the initial state in PLATO then $\forall \psi \in D(\mathcal{BF})$ $\mathsf{u} \models_{\Phi} \psi$ iff $\mathsf{v} \models_{\Gamma} \psi$.

Proposition 3 (*PLATO actions correctness w.r.t.* $m\mathcal{A}^{\rho}$)
Given an action instance $\mathsf{a} \in D(\mathcal{AI})$ and two possibilities $\mathsf{u}, \mathsf{v} \in D(\mathcal{S})$ such that $\forall \psi \in D(\mathcal{BF})$ $\mathsf{u} \models_{\Phi} \psi$ iff $\mathsf{v} \models_{\Gamma} \psi$ then $\forall \psi \in D(\mathcal{BF})$ $\Phi(\mathsf{a}, \mathsf{u}) \models_{\Phi} \psi$ iff $\Gamma(\mathsf{a}, \mathsf{v}) \models_{\Gamma} \psi$.

The complete proofs are provided in the supplementary documents that are available at `http://clp.dimi.uniud.it/sw/`. This results allowed us to verify the empirical correctness of the planner EFP 2.0. In all of the conducted tests, the two planners exhibited the same behaviour. In the same way, PLATO can be used to verify empirically the correctness of any multi-agent epistemic planner that is based on a semantics equivalent to the one of $m\mathcal{A}^{\rho}$. Finally, as the *plan existence problem* in the MEP setting is *undecidable* (Bolander and Andersen 2011), all the planners that reason on DEL are *incomplete*. Since infinitely many e-states could be potentially generated during a planning process, in general both EFP 2.0 and PLATO are unable to determine if a solution for a planning problem exists (even when checking for already visited states).

7 Conclusions and Future Works

In this paper we presented a multi-agent epistemic planner implemented in ASP. The implementation of MEP in a declarative language involves various advantages. First, the reduced size of the program allows a better readability of the code, allowing a much easier approach to MEP. Second, code maintainability is simpler and, third, modifications on the semantics of $m\mathcal{A}^{\rho}$ can be manageably implemented. Specifically, if new operators or actions types are added to $m\mathcal{A}^{\rho}$ (*e.g.*, concepts such as: *trust*, *lies* or *inconsistent beliefs*), it would suffice to add or modify a small number of rules. Ultimately, the extent of the code adaptation would be significantly lower w.r.t. the imperative approaches.

We were able to exploit several ASP features, such as the multi-thread parallelisation and the different solving configurations. Approaching MEP through declarative programming will also allow automated epistemic reasoning to benefit from the constant enhancement of ASP solvers' efficiency. These features, together with an efficient grounding, allowed us to achieve comparable results w.r.t. EFP 2.0. ASP also allows to find *all* the solutions of a planning instance without any addition to the code. The formal

https://doi.org/10.1017/S1471068420000289 Published online by Cambridge University Press

verification of the correctness of PLATO (Section 6) permitted us to empirically verify the correctness of EFP 2.0 by comparing the obtained plans on both solvers.

As future works, we plan to improve PLATO in several ways. First, we plan to enhance the entailment, by defining different entailment rules for different formulae types (*e.g.*, executability conditions, actions effect conditions, etc.). This will impact on both grounding and solving efficiency. We also plan a more efficient design of the update of the repetition values through hashing functions or bit maps; this would limit the growth of the repetition to a polynomial rate.

Second, we plan to implement some heuristics, such as choosing to perform the action that leads to the satisfaction of the higher number of goal conditions, so as to improve the computational results. Finally, we plan to exploit PLATO to formally prove that $m\mathcal{A}^*$ and $m\mathcal{A}^\rho$ are semantically equivalent. This would provide a much stronger result w.r.t. the one proved by Fabiano et al. in previous work (Fabiano et al. 2020).

Acknowledgments. The authors wish to thank Martin Gebser and Roland Kaminski for their suggestions on the ASP encoding and the anonymous Reviewers for their comments that allowed to improve the presentation. The research is partially supported by Indam GNCS, by Uniud project ENCASE, and by NSF grants 1914635, 1833630, and 1345232.

References

Aczel, P. 1988. Non-well-founded sets. CSLI Lecture Notes, 14, Stanford University, Center for the Study of Language and Information.

Baral, C., Gelfond, G., Pontelli, E., and Son, T. C. 2015. An action language for multi-agent domains: Foundations. *CoRR abs/1511.01960.*

Baral, C., Gelfond, G., Son, T. and Pontelli, E. 2010. Using answer set programming to model multi-agent scenarios involving agents' knowledge about other's knowledge. In Proc of AAMAS, Vol.1, 259-266.

Bolander, T. and Andersen, M. 2011. Epistemic planning for single-and multi-agent systems. *Journal of Applied Non-Classical Logics 21,* 1, 9–34.

Castellini, C., Giunchiglia, E., and Tacchella, A. 2001. Improvements to sat-based conformant planning. In *Proceedings of 6th European Conference on Planning (ECP-01).*

de Weerdt, M., Bos, A., Tonino, H., and Witteveen, C. 2003. A resource logic for multi-agent plan merging. *Ann. Math. Artif. Intell. 37,* 1-2, 93–130.

de Weerdt, M. and Clement, B. 2009. Introduction to planning in multiagent systems. *Multiagent and Grid Systems 5,* 4, 345–355.

Dovier, A. 2015. Logic programming and bisimulation. In *ICLP*, M. D. Vos, T. Eiter, Y. Lierler, and F. Toni, Eds. CEUR vol. 1433.

Dovier, A., Formisano, A., and Pontelli, E. 2013. Autonomous agents coordination: Action languages meet CLP(FD) and LINDA. *Theory Pract. Log. Program. 13,* 2, 149–173.

Durfee, E. H. 1999. Distributed continual planning for unmanned ground vehicle teams. *AI Magazine 20,* 4, 55–61.

Fabiano, F., Burigana, A., Dovier, A., and Pontelli, E. 2020. EFP 2.0: A multi-Agent Epistemic Solver with Multiple e-State Representations. *Proc of ICAPS*, 101–109.

Fabiano, F., Riouak, I., Dovier, A., and Pontelli, E. 2019. Non-well-founded set based multi-agent epistemic action language. In Proc of CILC, CEUR vol. 2396, 242–259.

Fagin, R., Halpern, J., Moses, Y., and Vardi, M. 2004. *Reasoning about knowledge.* MIT press.

https://doi.org/10.1017/S1471068420000289 Published online by Cambridge University Press

Gebser, M., Kaminski, R., Kaufmann, B., and Schaub, T. 2019. Multi-shot asp solving with clingo. *Theory and Practice of Logic Programming 19*, 27–82.

Gelfond, M. and Lifschitz, V. 1988. The stable model semantics for logic programming. In Proc of ICLP/ILPS, R. A. Kowalski and K. A. Bowen, Eds. MIT Press, 1070–1080.

Gelfond, M. and Lifschitz, V. 1993. Representing action and change by logic programs. *J. Log. Program. 17,* 2/3&4, 301–321.

Gelfond, M. and Lifschitz, V. 1998. Action languages. *Electron. Trans. Artif. Intell. 2*, 193–210.

Gerbrandy, J. and Groeneveld, W. 1997. Reasoning about information change. *Journal of Logic, Language and Information 6,* 2, 147–169.

Goldman, C. V. and Zilberstein, S. 2004. Decentralized control of cooperative systems: Categorization and complexity analysis. *JAIR 22*, 143–174.

Huang, X., Fang, B., Wan, H., and Liu, Y. 2017. A general multi-agent epistemic planner based on higher-order belief change. In Proc of *IJCAI*, 1093–1101.

Kominis, F. and Geffner, H. 2015. Beliefs in multiagent planning: From one agent to many. In *Proc of ICAPS*. 147–155.

Kripke, S. A. 1963. Semantical considerations on modal logic. *Acta Philosophica Fennica 16,* 1963, 83–94.

Le, T., Fabiano, F., Son, T. C., and Pontelli, E. 2018. EFP and PG-EFP: Epistemic forward search planners in multi-agent domains. In *Proc of ICAPS*, 161–170.

Löwe, B., Pacuit, E., and Witzel, A. 2011. Del planning and some tractable cases. In *International Workshop on Logic, Rationality and Interaction*. Springer, 179–192.

Son, T. C., Pontelli, E., Baral, C., and Gelfond, G. 2014. Finitary S5-theories. In *European Workshop on Logics in Artificial Intelligence*. Springer, 239–252.

Son, T. C., Tu, P. H., Gelfond, M., and Morales, A. R. 2005. An approximation of action theories of and its application to conformant planning. In Proc of *LPNMR*, Springer, 172–184.

Van Der Hoek, W. and Wooldridge, M. 2002. Tractable multiagent planning for epistemic goals. In *Proc of AAMAS: part 3*. ACM, 1167–1174.

Van Ditmarsch, H., van Der Hoek, W. and Kooi, B. 2007. Dynamic epistemic logic. Volume 337, Springer Science & Business Media.

https://doi.org/10.1017/S1471068420000289 Published online by Cambridge University Press