Life is what happens to us while we are making other plans.

— Allen Saunders
Reader's Digest, January 1957

5 Comprehensive Multi-Agent Epistemic Planners

Contents

5.1 Background

As already mentioned in Section 1.2, one of the planning community researchers' most important objectives is to develop automated tools to solve various planning problems. These tools, known as planners or solvers (Definition 1.7), concretize all the theoretical studies to solve problems using all the studied techniques. In what follows we will introduce two solvers, for Multi-agent Epistemic planning problems,

that incorporate all the theoretical innovations described in the previous chapters. These planners share the same objective, *i.e.*, solving problems in the multi-agent epistemic setting, but are implemented using different programming paradigms. We will firstly explore these two paradigms and their differences. We will then illustrate the two planners, describing their design along with some experimental evaluations.

5.1.1 Imperative and Declarative Programming

In this brief introduction, we will present two well-known programming paradigms, *i.e.*, *imperative* and *declarative*. These two approaches are widely studied in the computer science community and, for the sake of readability, we will assume that the reader is familiar with such concepts. That is why what follows is just a high-level characterization of the topics. For a more complete discussion on these two paradigms, we address the reader to the work by Fahland et al. [2009].

Imperative Programming

Let us start by describing the more "classical" of the two paradigms: imperative programming. This approach is, in fact, the one that arose alongside the computers. The programs modeled following this paradigm are made of a series of precise instructions. These instructions, or commands, are executed sequentially and are deterministic and, generally, read or write values stored in the computer memory. An easy, and yet clear, transposition of an imperative program in our everyday life is a recipe. A recipe, in fact, is comprised of a series of instructions that the reader must follow if she/he wants to obtain the desired result. This means that an imperative program is similar to our way of giving or executing instructions, therefore it is only natural that this approach is the mostly adopted one. Nevertheless, while recipes' instructions do not need to be too specific, computer commands must be very detailed given, that the machines lack any sense of "interpretation". This means that complex imperative programs might be comprised of a very large sequence of commands, making it hard to: debug, maintain, explain, adapt, and so on. Examples,

among many others, of imperative programming languages are C++ [Stroustrup, 2013], Python [Van Rossum and Drake, 2009], and Java [Arnold et al., 2005].

Declarative Programming

A different type of approach with respect to the one aforementioned is known as declarative programming. Among the declarative programming paradigms, one of the most mature and important in AI is *logic* programming. This paradigm stems from first-order logic and it has gained a lot of attention in recent years. The declarative approach aims to solve a problem by describing it and its rules, rather than explicitly stating the instructions to follow. This makes declarative programming more suited for all those situations where the problem definition, its constraints, and the structure of its solution are known, and defining a procedural algorithm may require great efforts. Examples of such problems are the *n-queens problem* [Bowtell and Keevash, 2021] or the well-known Sudoku [Hanson, 2021]. In particular, in this thesis, we will make use of the language known as Answer Set Programming [Lifschitz, 2008], or ASP for short, introduced next.

Answer Set Programming 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 *ground*. 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 with respect to 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 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 defined *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.

5.2 EFP: an Epistemic Forward Planner

The first system that this thesis will present is a solver, called EFP, designed following the imperative programming paradigm. In particular, the solver, that can be found at `https://github.com/FrancescoFabiano/EFP`, is fully developed in

C++ [Stroustrup, 2013]. EFP is a *general and comprehensive epistemic forward solver* that can solve problems defined in $m\mathcal{A}^\rho$. Moreover, thanks to the introduction of agents' attitudes and other capabilities (explored later in this chapter), the planner will allow the users to tailor actions in whichever fashion they prefer without having to worry about tedious and intricate effects definitions. This will help in formalizing new scenarios in which agents can reason while considering belief relations with concepts such as lies, misconceptions, trust, and so on and where different groups of agents react differently to the actions.

5.2.1 The Overall Architecture

The overall architecture of EFP is given in Algorithm 1 even if it is not different from the standard algorithm implemented by search-based planners. Nonetheless, EFP has a modular organization that facilitates modifications and extensions. The key modules of EFP are *(i)* a pre-processor; *(ii)* initial e-state computation; and *(iii)* a search engine.

(i) *Pre-processor*: This module is responsible for parsing the planning problem description, setting up the planning domain, which includes the list of agents, the list of actions, the rules for computing frames of reference, and the list of fluent literals. This module is also responsible for the initialization of necessary data structures (*e.g.*, e-states) and executes some transformations.

(ii) *Initial e-state computation*: This module is responsible for computing the set of initial e-states. Under the assumption that the initial state description encodes a finitary S5-theory in the sense of Son et al. [2014], we know that the set of initial e-states is finite (up to bisimulation). This module implements the algorithm given in Son et al. [2014] for computing the aforementioned set of initial e-states.

(iii) *Search engine*: This module is responsible for computing a solution. EFP implements different research strategies such as breadth-first, depth-first, and best-first search (Algorithm 1), that can be selected by the user to solve

the desired problem. The heuristics used by EFP, when best-first search is selected, are presented in the next section. Finally, this search engine is able "to reason" on both the two e-states representations discussed in the previous chapters, *i.e.*, Kripke structures and possibilities.

Algorithm 1: EFP Best-First Search

Input : A planning problem $P = \langle \mathcal{F}, \mathcal{AG}, A, O, s_0, \phi_g \rangle$
Output : A solution for P if exists; failed otherwise

Compute the initial e-state given s_0: $(\mathcal{M}_i, W_i)$
Initialize a priority queue $q = [(\{(\mathcal{M}_i, W_i)\}, [])]$
while q *is not empty* **do**
  $(\Omega, plan) = dequeue(q)$
  if $(\mathcal{M}, W_d) \models \phi_g$ *for every* $(\mathcal{M}, W_d) \in \Omega$ **then**
    return *plan*
  end
  for *action* a *executable in every* $(\mathcal{M}, W_d)$ *in* Ω **do**
    Compute $\Omega' = \bigcup_{(\mathcal{M}, W_d) \in \Omega} \Phi(a, (\mathcal{M}, W_d))$
    Compute heuristics and insert $(\Omega', plan \circ a)$ into q
  end
end
return failed

5.2.2 EFP 2.0

The solver EFP was firstly introduced by Le et al. [2018] as the first epistemic planner able to deal with unlimited nested belief formulae and dynamic common knowledge. This original planner, which we will identify with EFP 1.0, was based on the action language $m\mathcal{A}^*$ and, therefore, used Kripke structures as e-states representation. The planner allowed for both breadth-first and best-first searches. The heuristic used in the latter was the so-called *Epistemic Planning Graph* that allowed reasoning on partial Kripke structures to derive the score of the various e-states. For more details on EFP 1.0 and on the Epistemic Planning Graph we address the reader to the work by Le et al. [2018].

In this work, we present an updated version of the planner presented by Le et al.. For clarity, we will call such updated planner EFP 2.0. This new solver

redesigned every element of EFP 1.0 to introduce multiple e-states representations and, therefore, multiple transition functions. On the other hand, our implementation keeps the same modular structure of EFP 1.0. The planning process executed by EFP 2.0 is, primarily, a *breadth-first search* with duplicate checking. Other types of searches have been also implemented and will be presented later in this chapter. Let us note that the computation of the initial state is not a trivial task in MEP. In particular, given a belief formula φ_{ini} it is, in general, possible to generate infinite e-states that respect φ_{ini}. As mentioned, to overcome this problem EFP 1.0 imposes that the initial state description should be a *finitary* ***S5****-theory* [Son et al., 2014]. In EFP 2.0 we still require the initial description to be a finitary **S5**-theory but we allow φ_{ini} to be less specific. In particular, without going into details of finitary **S5**-theories, whenever a fluent literal `f` is not considered by φ_{ini}, EFP 2.0 assumes that is common knowledge between all the agents that `f` is unknown.

Another remark that has to be done is about the e-states. EFP 2.0 has a "templatic" e-state definition. This means that each solving process can be executed using the desired e-state representation with its relative transition function. Currently, EFP 2.0 implements two e-states representations, *i.e.*, Kripke structures and possibilities, and diverse transition functions that can be selected to solve the given problem:

- the one introduced in Baral et al. [2015] (for Kripke structures);
- the transition function for possibilities introduced in Definition 2.12 that emulates the behavior of $m\mathcal{A}^*$;
- the enriched possibilities update that allows for agents to be characterized with attitudes presented in Definition 4.4; and
- an experimental transition function for possibilities that allows for user-defined update models to be adopted. We will analyze this configuration later in this section.

Another important concept that EFP 2.0 integrates is the Kripke structures size reduction. We implemented two algorithms, following the works by Paige and Tarjan [1987], Dovier et al. [2004], that starting from a generic Kripke structure compute its bisimilar, and therefore semantically equivalent, correspondent with minimal size.

Finally, EFP 2.0 introduces the concept of "*already visited e-state*". Excluding the already visited states during the planning is a common practice and it is done in the majority of the solving processes. Nevertheless, EFP 1.0 did not implement the comparison of visited states. That is because comparing two e-states is not as trivial as comparing, for instance, two sets of fluent literals. In fact, being each e-state in $m\mathcal{A}^*$ a Kripke structure, comparing two e-states means checking for *isomorphism* or *bisimulation* between them. That is why in EFP 1.0 the comparison for already visited states was left as future development. On the other hand, with possibilities, the equality check should be faster since, thanks to the non-well-foundeness, we can collapse each possibility in a small system of equations and exploit the already calculated possibilities information. That is why in EFP 2.0 we implemented the visited e-state check initially for possibilities and later for Kripke structures.

5.2.3 Experimental Evaluation

In this paragraph we compare the new multi-agent epistemic planner EFP 2.0 with, to the best of our knowledge, the only other comprehensive multi-agent epistemic solver in literature, *i.e.*, the planner presented in Le et al. [2018]. All the experiments were performed on a 3.60GHz Intel Core i7-4790 machine with 32GB of memory.

From now on, to avoid unnecessary clutter, we will make use of the following notations:

- L to indicate the (optimal) length of the plan;
- `WP` to indicate that the solving process returned a `Wrong Plan`;
- `TO` to indicate that the solving process did not return any solution before the timeout (25 minutes);

- EFP 1.0 to denote the Breadth-First search planner presented in Le et al. [2018]. We chose the Breadth-First solver because we wanted to focus on the basis of the solving process so that all the future optimizations could benefit from this research.

- K-MAL to identify our solver while using Kripke structures as e-state representation and the transition function of Baral et al. [2015].

- K-BIS to identify our solver while using Kripke structures as e-state representation and the algorithm to find the coarsest refinement, presented in Paige and Tarjan [1987], to minimize the e-states size. We also tried to compact the e-states using the algorithm presented in Dovier et al. [2004] but the performances were almost identical. This is probably because the Kripke structures we are considering are relatively small in size.

- P-MAR to identify our solver while using possibilities as e-state with the transition function introduced in Definition 2.12.

All the configurations K-MAL, K-BIS, and P-MAR check for already visited states. To indicate the same configurations without the visited states check we will use K-MAL-NV, K-BIS-NV, and P-MAR-NV.

We evaluate EFP 2.0 on benchmarks collected from the literature [Kominis and Geffner, 2015, Huang et al., 2017]. In particular, these domains are:

(i) Collaboration and Communication (**CC**). In this domain, $n \geq 2$ agents move along a corridor with $k \geq 2$ rooms in which $m \geq 1$ boxes can be located. Whenever an agent enters a room, she can determine if a certain box is in the room. Moreover, agents can communicate information about the boxes' position to other *attentive* agents. The goals consider agents' positions and their beliefs about the boxes (Table 5.1).

(ii) Selective Communication (**SC**). **SC** has $n \geq 2$ agents that start in one of the $k \geq 2$ rooms in a corridor. An agent can tell some information and all the agents in her room or the neighboring ones can hear what was told.

L	EFP 1.0	K-MAL	K-BIS	P-MAR	EFP 1.0	K-MAL	K-BIS	P-MAR
	CC_1: $\|\mathcal{AG}\| = 2$, $\|\mathcal{F}\| = 10$, $\|\mathcal{A}\| = 16$				**CC**_3: $\|\mathcal{AG}\| = 3$, $\|\mathcal{F}\| = 14$, $\|\mathcal{A}\| = 24$			
3	.08	.05	.08	**.02**	.12	.07	.13	**.03**
4	.16	.09	.16	**.03**	.56	.31	.54	**.10**
5	1.31	.79	1.14	**.16**	6.55	3.25	4.89	**.60**
6	6.99	3.58	4.42	**0.64**	25.11	9.09	12.66	**1.71**
7	49.44	15.95	16.06	**2.61**	TO	92.37	142.06	**12.37**
	CC_2: $\|\mathcal{AG}\| = 2$, $\|\mathcal{F}\| = 14$, $\|\mathcal{A}\| = 28$				**CC**_4: $\|\mathcal{AG}\| = 3$, $\|\mathcal{F}\| = 14$, $\|\mathcal{A}\| = 42$			
3	.31	.21	.37	**.07**	.62	.54	.81	**.15**
4	1.54	.98	1.77	**.26**	3.22	2.84	5.40	**.87**
5	22.14	12.55	18.80	**1.68**	104.97	106.02	152.38	**7.41**
6	171.19	72.92	102.97	**7.71**	473.03	246.08	313.70	**25.47**
7	TO	437.91	592.48	**38.81**	TO	TO	TO	**174.67**

Table 5.1: Runtimes for the Collaboration and Communication (**CC**) domain.

Every agent is free to move from one room to its adjacent. The goals usually require some agents to know certain properties while other agents ignore them (Figure 5.1).

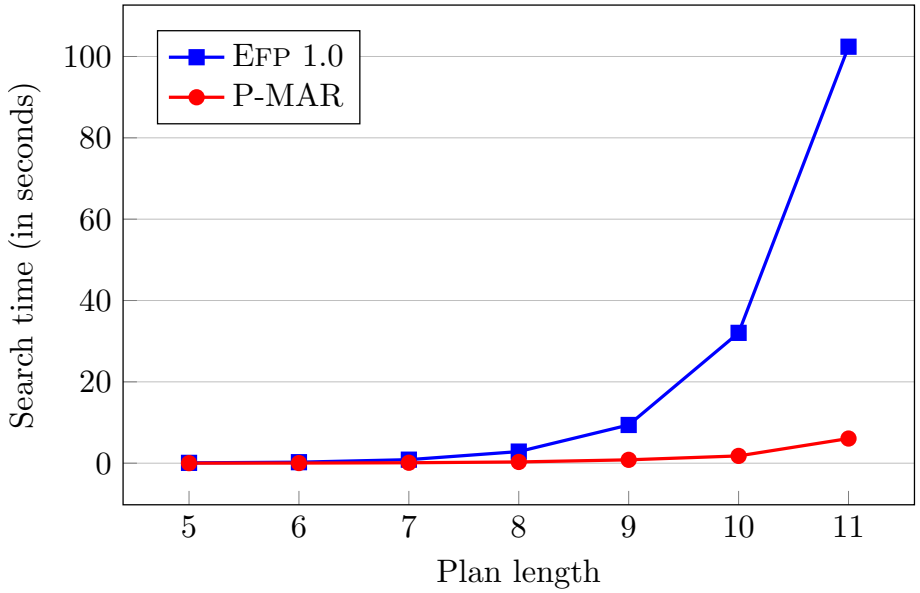


Figure 5.1: Comparison between EFP 1.0 and P-MAR on **SC** instances with $k = 11$ rooms and $|\mathcal{AG}| = 9$.

(iii) Grapevine (**GR**). $n \geq 2$ agents are located in $k \geq 2$ rooms. An agent can move freely to each other room and she can share a "secret" with the agents that are in the room with her. This domain supports different goals, from

sharing secrets with other agents to having misconceptions about agents' beliefs (Table 5.2).

Grapevine										
$\lvert\mathcal{AG}\rvert$	$\lvert\mathcal{F}\rvert$	$\lvert\mathcal{A}\rvert$	L	EFP 1.0	K-MAL-NV	K-MAL	K-BIS-NV	K-BIS	P-MAR-NV	P-MAR
3	9	24	2	WP	.09	.09	.19	.20	.03	**.02**
			4	WP	9.19	8.13	13.54	12.76	1.34	**1.25**
			5	WP	94.49	75.32	111.38	84.46	8.67	**7.71**
			6	WP	372.64	278.93	398.10	232.54	27.63	**20.26**
4	12	40	2	WP	1.85	1.786	1.95	2.08	**.17**	.18
			4	WP	403.11	274.53	178.52	111.38	13.49	**7.31**
			5	WP	TO	TO	TO	775.63	123.54	**36.54**
			6	WP	TO	TO	TO	TO	427.97	**108.64**

Table 5.2: Runtimes for the Grapevine (**GR**) domain. We compare the configurations with and without the visited e-states check. EFP 1.0 errors are caused by a wrong initial e-state generation.

(iv) Coin in the Box (**CB**). This domain is firstly presented in Planning Domain 2.1 we will still provide a brief description of it. $n \geq 3$ agents are in a room where in the middle there is a box containing a coin. None of the agents know whether the coin lies heads or tails up and the box is locked. One agent has the key to open the box. The goals usually consist in some agents knowing whether the coin lies heads or tails up while other agents know that she knows, or are ignorant about this (Table 5.3).

CB with $\lvert\mathcal{AG}\rvert = 3$, $\lvert\mathcal{F}\rvert = 8$, $\lvert\mathcal{A}\rvert = 21$				
L	EFP 1.0	K-MAL	K-BIS	P-MAR
2	.003	.003	.006	**.001**
3	.048	.077	.097	**.016**
5	WP	5.546	1.438	**.367**
6	WP	108.080	14.625	**2.932**
7	WP	317.077	38.265	**6.996**

Table 5.3: Runtimes for the Coin in the Box (**CB**) domain.

(v) Assembly Line (**AL**). In this problem, there are two agents, each responsible for processing a different part of a product. Each agent can fail in processing her part and can inform the other agent of the status of her task. Two agents decide to *assemble* the product or *restart*, depending on their knowledge about

the product status. The goal in this domain is fixed, *i.e.*, the agents must assemble the product, but what varies is the *depth* of the belief formulae used as executability conditions (Table 5.4).

AL with $\|\mathcal{AG}\| = 2$, $\|\mathcal{F}\| = 4$, $\|\mathcal{A}\| = 6$				
d	EFP 1.0	K-MAL	K-BIS	P-MAR
2	.43	.32	.42	**.07**
4	.96	.75	.64	**.11**
6	26.20	27.85	13.51	**2.44**
8	TO	TO	883.87	**150.92**
C	.44	.32	.43	**.08**

Table 5.4: Runtimes for the Assembly Line (**AL**) domain. The last row identifies the instance where the executability conditions are expressed through common belief.

All our experiments (Tables 5.1 to 5.4, Figure 5.1) show that EFP 2.0, if used with its fastest configuration P-MAR, performs significantly better than EFP 1.0. We believe that these results derive from several factors.

First and foremost the choice of using possibilities as e-states and $m\mathcal{A}^{\rho}$ as action language ensured that every e-state generated during the planning process had always smaller or equal size with respect to the same state generated in EFP 1.0. In particular, EFP 1.0, generating e-states with non-minimal size, introduces extra (always increasing) overhead at each action application with respect to EFP 2.0. This is illustrated in Table 5.5 where the number of worlds and edges generated by EFP 1.0 & K-MAL and K-BIS & P-MARafter executing an action instances sequence is compared. Let us note that Table 5.5 is graphically rendered in Figure 2.12.

Moreover, the implementation of P-MAR exploits already calculated e-states information when it creates new ones reducing even more the e-states generation time (this factor is not considered in Table 5.5). From our results, it is clear that EFP 1.0 and P-MAR perform similarly on very small instances of the problems but as soon as the problem grows the two solvers have different behaviors. In fact, while EFP 1.0 search time increases very rapidly P-MAR stays relatively stable. That is because when the problems become more complex the planner, generally, has to generate more e-states. Regarding the other configurations of EFP 2.0, namely

CB with $\lvert\mathcal{AG}\rvert = 3$, $\lvert\mathcal{F}\rvert = 8$, $\lvert\mathcal{A}\rvert = 21$				
L	|Worlds|		|Edges|	
	EFP 1.0 & K-MAL	K-BIS & P-MAR	EFP 1.0 & K-MAL	K-BIS & P-MAR
1	6	6	36	36
2	12	**9**	70	**53**
3	24	**14**	138	**82**
4	48	**19**	274	**111**
5	85	**23**	465	**131**
6	159	**31**	847	**171**
7	273	**38**	1409	**201**
8	468	**45**	2435	**231**
9	819	**52**	4361	**261**
10	1461	**59**	8037	**291**

Table 5.5: Comparison of the e-states' size, in terms of worlds (left) and edges (right), generated by the various solving processes on the Coin in the Box (**CB**) domain.

K-MAL and K-BIS, we note that they generally outperform EFP 1.0. Nevertheless, in some cases (Tables 5.1 and 5.4), we note some exceptional peaks in these configuration's performances. These peaks are the results of *(i)* the use of the visited-state check that in some configurations may add an extra overhead that in EFP 1.0 was not present; and *(ii)* a less optimized *entailment-check* function, with respect to EFP 1.0, in the configurations of EFP 2.0 that are based on Kripke structures. A remark has to be done on the K-BIS configuration. From the results (Tables 5.1 to 5.4), it is clear how this configuration, even if executes the solving process on minimal-sized e-states, it is still outperformed by P-MAR. The reasons for this are essentially two: *(i)* thanks to their non-well-founded nature possibilities allow re-using already generated information during the planning process; and *(ii)* the use of external algorithms to minimize the size of the e-states introduces an extra overhead with respect to P-MAR.

Another important factor that makes EFP 2.0 faster than EFP 1.0 is the concept of visited e-states. As we can see in Table 5.2 the planner takes advantage of this check even when the e-states are represented as Kripke structures. The fact that the visited-state check increases the performances of EFP 2.0 proves that, even if

this check relies on "heavy" algorithms, the epistemic planning process benefits from the elimination of the duplicates.

Finally, the complete refactoring of the code helped us to implement a more efficient solver. In fact, even if EFP 2.0 is based on EFP 1.0, the remodeling of the solver allowed us: *(i)* to correct bugs related to the initial e-state generation (Table 5.2) and to the transition function (Table 5.3); and *(ii)* to optimize the code. This optimization is reflected by the comparison between K-MAL and EFP 1.0. In fact, these two configurations both use Kripke structures as e-states and implement $m\mathcal{A}^*$ [Baral et al., 2015]. Nevertheless, K-MAL generally outperforms EFP 1.0 as shown in Table 5.1.

Alternative Transition Functions

Enriched $m\mathcal{A}^\rho$ Update As mentioned above, EFP 2.0, besides implementing the language $m\mathcal{A}^\rho$, allows the user to exploit two different transition functions. The first one is fully described in Chapter 4, in particular in Definition 4.4. This transition function allows to describe agents with several attitudes and to solve domains where concepts such as trust and lies are involved. The performances of the planner while using this transition function are almost identical with respect to P-MAR on domains that can be solved by both configurations. On the other hand, domains where this transition function "full potential" is required cannot be solved by other configurations. That is why, for the sake of readability, we will not report any experimental comparison for the transition function of Definition 4.4 and other EFP 2.0 configurations. Nonetheless, the enriched semantics of $m\mathcal{A}^\rho$ has been implemented in EFP 2.0 that is now able to tackle families of problems that consider complex aspects such as doxastic reasoning, lying agents, faulty perception, etc. Let us note that Figures 4.2 to 4.6 are automatically generated by the planner and, therefore, constitute examples of execution.

One of our main interests is to compare the expressive power of the new semantics with other approaches in the literature. To this end, we tested a small variation of the *Grapevine* domain (that, even though it does not fully explore the newly

introduced concepts, comprises elements such as misconception and lies) on both EFP 2.0 and the planner RP-MEP introduced in Muise et al. [2015]. The latter firstly encodes an MEP problem into a classical planning problem. Next, the solving phase is handled by a classical planner. The results are reported in Table 5.6. The

GR with $\lvert\mathcal{AG}\rvert = 4$, $\lvert\mathcal{F}\rvert = 16$, $\lvert\mathcal{A}\rvert = 32$			
d	L	EFP 2.0	RP-MEP
2	3	21.58 s	**9.33** s
4		**21.58** s	3189.05 s
≥ 5		**21.58** s	`Time-Out`
2	4	409.53 s	**9.41** s
4		**409.53** s	3201.13 s
≥ 5		**409.53** s	`Time-Out`

Table 5.6: Runtimes, in seconds, of the Grapevine (**GR**) domain with a `Time-Out` of 1800 s.

comparison shows how RP-MEP outperforms EFP 2.0 when the formulae depth parameter d is small, due to the efficiency of classical planners. However, since the size of the encoded classical problem is exponential with respect to d, increasing the depth of DEL formulae results in efficiency loss on the former solver. On the other hand, EFP 2.0 scales better when the value of d is increased since the space required from the latter solver does not depend on such parameter.

Custom Update Models Finally, let us introduce a configuration of EFP 2.0 that takes advantage of diverse factors, that is possibilities, update models, and agents' attitudes. While this configuration is not directly derived by some theoretical innovation, it combines the diverse capabilities of the languages to increase the functionalities of the planner. In particular, this configuration allows to define *custom update models* (Definition 2.2) for Possibilities. These custom update models allow the user to specify all sorts of behaviors for the actions, making it possible to capture all the variations in the belief update in epistemology. This level of customization permits to confront several theories on how the agents' beliefs must be updated when, for example, in presence of lies, stubbornness, trust ignorance, and so on. The e-states update follows the schema presented in Definition 2.3. The

specification of custom update models is made through a PDDL-like syntax as we can see in Listing 5.1. In Listing 5.1, we show how custom event models could be used to represent the transition function presented in Figure 2.2 while, in Listing 5.2, we provide an example of instantiated actions in $m\mathcal{A}^{\rho}$. Let us note that, for the sake of simplicity, we unified sensing and announcement under the action type *epistemic*.

The specification starts with the definition of the single events (the squares nodes in Figure 2.2) in Lines 1-26. Each event, besides specifying its unique **id**, is also characterized by **preconditions** and **postconditions**. Both conditions can be a conjunction of the following, possibly negated, meta-values: `$act_eff$`, `$act_pre$` and `none`. `$act_eff$` and `$act_pre$` are proxies for the action's (from which we are deriving the instantiated update model) effects and preconditions, respectively. `none`, on the other hand, is used to explicitly tell that the conditions are empty.

Next, in Line 29, the observability groups are presented. These groups represent all the possible sets in which agents may belong. An example of this is shown in Lines 6-8 and 16-18 of Listing 5.2 where the observability of the agents `a` and `b` is defined. The order in which these statements are written matters, as the solver returns as observability group the first that has its conditions verified—the truth of such conditions depends on the specific e-state on which they are checked.

Finally, from Line 31 of Listing 5.1, the structure of the event model is described. Each model, identified by a unique `id`, specifies which events considers and, among them, which is the pointed one (the bold nodes in Figure 2.2). Moreover, the labeled edges of the model are also specified with a syntax of the form `(outgoing_node, incoming_node, label)`. Listing 5.2 provides an example of event model instantiation. To be more precise, the action `open_a` (Lines 3-8), follows the model `Ontic` with precondition `has_key_a` and effects `opened`. This means that the event model of this action is comprised of event `4` with postcondition `opened` and of event `3`. Moreover, agent `a` always belongs to the `Fully` group while `b` could be in either `Fully` or `Oblivious` depending on the value of `looking_b`.

```
1  ***************Events definition*************
2
3   (event:
4    :id              (1)     *event "sigma-epistemic"
5    :precondition    ($act_eff$)
6    :postcondition   (none)
7   )
8
9   (event:
10   :id              (2)     *event "tau"
11   :precondition    (not($act_eff$))
12   :postcondition   (none)
13  )
14
15  (event:
16   :id              (3)     *event "epsilon"
17   :precondition    (none)
18   :postcondition   (none)
19  )
20
21  (event:
22   :id              (4)     *event "sigma-ontic"
23   :precondition    (none)
24   :postcondition   ($act_eff$)
25  )
26
27 ****************Observability Groups***************
28
29  (obs_groups: {Fully;Partially;Oblivious})
30
31 **************Event Models Definition***************
32
33  (model:
34   :id       (Epistemic)     *Sensing - Annoucement Action
35   :events   {1;2;3}
36   :pointed  (1)
37   :edges    {(1,1,Fully)(2,2,Fully)(3,3,Fully)
38              (1,1,Partially)(2,2,Partially)(3,3,Partially)
39              (1,2,Partially)(2,1,Partially)
40              (1,3,Oblivious)(2,3,Oblivious)(3,3,Oblivious)}
41  )
42
43  (model:
44   :id       (Ontic)
45   :events   {4;3}
46   :pointed  (4)
47   :edges    ((4,4,Fully)(3,3,Fully)(4,3,Oblivious)(3,3,Oblivious))
48  )
```

Listing 5.1: The transition function of $m\mathcal{A}^*$ described as custom update template

```
1  **************Ontic action: open_a**************
2
3  executable open_a if has_key_a;
```

```
4  open_a has_effects opened;
5  open_a has_type Ontic;
6  a in_group Fully of open_a;
7  b in_group Fully of open_a if looking_b;
8  b in_group Oblivious of open_a;
9
10
11 **************Epistemic action: shout_tail_a**************
12
13 executable shout_tail_a if B(a,tail), tail;
14 shout_tail_a has_effects tail;
15 shout_tail_a has_type Epistemic;
16 a in_group Fully of shout_tail_a;
17 b in_group Fully of shout_tail_a if looking_b;
18 b in_group Oblivious of shout_tail_a;
```

Listing 5.2: Examples of actions definitions with custom update models.

In Listing 5.1 we provided just an example of one possible custom update model. The advantage of this EFP 2.0 configuration is that is flexible enough to be adopted to test different update templates. This is useful, especially in combination with the planner capability of providing a graphical representation of the e-states[1], allowing to better understand how the new update template affects the e-states.

5.2.4 Optimizations and Alternative Search Strategies

Whilst the generality of the planner is of the utmost importance, reducing the search times, given the inherent complexity of MEP, is also a feature that is essential to our solver. That is why our final efforts were spent on developing more efficient data structures and processes of e-state updates along with some domain-independent heuristics and diverse search methods.

Code Optimizations

This section explores some of the efforts that allowed to optimize the performances of EFP 2.0. We will not explore in detail such optimizations as this would require a tedious explanation of all the involved data structures. On the other hand, we will provide an overall description of the changes followed by several tables that capture the results of these optimizations.

[1]Figures 4.2 to 4.6 are generated thanks to this functionality.

Thanks to the Valgrind profiler [Nethercote and Seward, 2007] we were able to identify which operations of EFP 2.0 spent most of the resources (time and memory). We noticed that, surprisingly, these operations were not complex tasks linked to epistemic reasoning but were related to `string` operations. We made use of `string` as internal ids for the various data structures of the planning process without realizing that such data type can bring severe overheads on C++ programs. That is why we restructured the planner so that it would make use of `dynamic_bitset`, provided by the library Boost [Schling, 2011], as internal ids instead of `strings`. This change affected most of the planner code but provided excellent results in terms of time and memory performances optimization as we can see in Tables 5.7 to 5.10 for the Time consumption, in seconds, and Tables 5.11 to 5.14 for the Memory consumption, in MB. Let us note that the changes only affected the underlying data structures and did not modify the search process. This means, that the two approaches shared the same search-tree topology when solving the same instance thus indicating that the improvements derived from the new data structure. As before, all the experiments were performed on a 3.60GHz Intel Core i7-4790 machine with 32GB of memory. Moreover, we will use:

- EFP 2.0: to indicate the configuration of EFP 2.0 before the conversion of the data structures;
- EFP 2.1: to indicate the optimized planner; and
- %: to indicate the percentage of resources "saved" by EFP 2.1 with respect to EFP 2.0.

CB with $\|\mathcal{AG}\| = 3, \|\mathcal{F}\| = 8, \|\mathcal{A}\| = 21$			
L	EFP 2.0	EFP 2.1	%
2	0.002	**0.001**	9.2
3	0.017	**0.015**	15.5
5	0.355	**0.249**	32.5
6	3.000	**2.283**	24.9
7	8.000	**6.233**	22.8

Table 5.7: Time consumption, in seconds, of EFP 2.0 and EFP 2.1 on the Coin in the Box (**CB**) domain.

AL with $\|\mathcal{AG}\| = 2, \|\mathcal{F}\| = 4, \|\mathcal{A}\| = 6$			
d	EFP 2.0	EFP 2.1	%
2	0.080	**0.047**	40.9
3	0.087	**0.051**	40.8
4	0.120	**0.069**	41.9
5	0.498	**0.297**	40.3
6	2.000	**1.450**	40.7
7	26.543	**15.690**	40.8
8	150.827	**90.982**	39.7
9	1689.322	**1003.420**	40.6
C	0.101	**0.055**	45.0

Table 5.8: Time consumption, in seconds, of EFP 2.0 and EFP 2.1 on the Assembly Line (**AL**) domain.

GR with $\|\mathcal{AG}\| = 3, \|\mathcal{F}\| = 9, \|\mathcal{A}\| = 24$				**GR** with $\|\mathcal{AG}\| = 4, \|\mathcal{F}\| = 12, \|\mathcal{A}\| = 42$			
L	EFP 2.0	EFP 2.1	%	L	EFP 2.0	EFP 2.1	%
2	0.0301	**0.0206**	31.4	2	0.221	**0.104**	52.6
3	0.202	**0.132**	34.0	3	1.452	**0.760**	49.6
4	1.374	**0.873**	36.5	4	10.490	**5.248**	49.9
5	9.125	**5.308**	41.8	5	72.228	**36.392**	49.6
6	22.216	**14.000**	36.7	6	198.431	**98.841**	50.2

Table 5.9: Time consumption, in seconds, of EFP 2.0 and EFP 2.1 on the Grapevine (**GR**) domain.

CC with $\|\mathcal{AG}\| = 2, \|\mathcal{F}\| = 10, \|\mathcal{A}\| = 16$				**CC** with $\|\mathcal{AG}\| = 2, \|\mathcal{F}\| = 14, \|\mathcal{A}\| = 28$			
L	EFP 2.0	EFP 2.1	%	L	EFP 2.0	EFP 2.1	%
3	0.022	**0.016**	28.7	3	0.081	**0.041**	48.5
4	0.035	**0.026**	25.2	4	0.280	**0.165**	43.3
5	0.195	**0.149**	23.3	5	2.371	**1.233**	48.9
6	0.807	**0.622**	22.7	6	9.990	**6.288**	37.1
7	3.311	**2.627**	20.7	7	48.810	**30.026**	38.5
CC with $\|\mathcal{AG}\| = 3, \|\mathcal{F}\| = 13, \|\mathcal{A}\| = 24$				**CC** with $\|\mathcal{AG}\| = 3, \|\mathcal{F}\| = 14, \|\mathcal{A}\| = 24$			
L	EFP 2.0	EFP 2.1	%	L	EFP 2.0	EFP 2.1	%
				3	0.166	**0.087**	47.9
4	0.119	**0.087**	26.9	4	0.846	**0.459**	45.7
5	0.864	**0.623**	27.9	5	14.980	**7.950**	46.8
6	3.000	**1.830**	27.5	6	47.330	**25.490**	46.1
7	23.453	**16.816**	25.7	7	394.871	**201.235**	49.0

Table 5.10: Time consumption, in seconds, of EFP 2.0 and EFP 2.1 on the Collaboration and Communication (**CC**) domain.

CB with $\|\mathcal{AG}\| = 3, \mathcal{F}\| = 8, \|\mathcal{A}\| = 21$			
L	EFP 2.0	EFP 2.1	%
2	0.0014	**0.0012**	9.2
3	**3.13**	3.75	-18.9
5	104.88	**38.97**	62.9
6	895.34	**387.53**	56.7
7	2635.73	**1303.63**	50.5

Table 5.11: Memory consumption, in MB, of EFP 2.0 and EFP 2.1 on the Coin in the Box (**CB**) domain.

SC with $\|\mathcal{AG}\| = 9, \|\mathcal{F}\| = 12, \|\mathcal{A}\| = 14$			
L	EFP 2.0	EFP 2.1	%
4	6.72	**5.75**	14.4
5	11.74	**7.78**	33.7
6	27.94	**13.85**	50.4
7	84.45	**34.87**	58.8
8	286.66	**100.63**	64.9
9	868.19	**313.31**	63.9
10	2833.88	**1004.46**	64.6
11	9242.77	**3246.91**	64.9

Table 5.12: Memory consumption, in MB, of EFP 2.0 and EFP 2.1 on the Selective Communication (**SC**) domain.

GR with $\|\mathcal{AG}\| = 3, \|\mathcal{F}\| = 9, \|\mathcal{A}\| = 24$				**GR** with $\|\mathcal{AG}\| = 4, \|\mathcal{F}\| = 12, \|\mathcal{A}\| = 42$			
L	EFP 2.0	EFP 2.1	%	L	EFP 2.0	EFP 2.1	%
2	12.35	**6.91**	44.1	2	93.38	**20.97**	77.5
3	63.52	**23.17**	63.5	3	698.63	**110.03**	84.2
4	427.15	**138.53**	67.6	4	6209.45	**962.95**	84.5
5	2812.83	**897.15**	68.1	5	10785.86	**5416.12**	49.8
6	7758.73	**2942.66**	62.1	6	10725.18	**5409.72**	49.6
7	7713.16	**5322.13**	31.0				

Table 5.13: Memory consumption, in MB, of EFP 2.0 and EFP 2.1 on the **Grapevine** (**GR**) domain.

CC with $\|\mathcal{AG}\| = 4, \|\mathcal{F}\| = 12, \|\mathcal{A}\| = 40$				**CC** with $\|\mathcal{AG}\| = 2, \|\mathcal{F}\| = 14, \|\mathcal{A}\| = 28$			
L	EFP 2.0	EFP 2.1	%	L	EFP 2.0	EFP 2.1	%
3	9.06	**4.25**	53.1	3	43.36	**10.06**	76.8
4	14.94	**7.04**	52.7	4	168.74	**29.17**	82.7
5	108.59	**41.96**	61.3	5	1871.65	**287.68**	84.6
6	541.29	**207.84**	61.6	6	11634.98	**1860.75**	84.0
7	2804.42	**1123.37**	59.9	7	12310.44	**4694.33**	61.8
CC with $\|\mathcal{AG}\| = 3, \|\mathcal{F}\| = 13, \|\mathcal{A}\| = 16$				**CC** with $\|\mathcal{AG}\| = 3, \|\mathcal{F}\| = 14, \|\mathcal{A}\| = 24$			
L	EFP 2.0	EFP 2.1	%	L	EFP 2.0	EFP 2.1	%
				3	91.31	**17.39**	80.9
4	65.33	**20.25**	69.0	4	564.14	**77.31**	86.2
5	770.84	**203.09**	73.6	5	13366.47	**1855.17**	86.1
6	2909.04	**767.28**	73.6	6	13270.54	**4497.85**	66.1
7	10559.78	**4750.69**	55.0	7	13289.85	**4492.25**	66.2

Table 5.14: Memory consumption, in MB, of EFP 2.0 and EFP 2.1 on the Collaboration and Communication (**CC**) domain.

Alternative Search Strategies and Heuristics

As mentioned before, planning on multi-agent epistemic domains is a very complex task. That is why even if optimizing the solving process is essential, only focusing on such a task may never allow epistemic planners to become tools suited for real-life scenarios.

Search Strategies For this reason, we decided to investigate alternative search strategies that may help in containing the resources needed to solve MEP problems. In particular, other than the standard *Breadth-First Search* (BFS), we enriched EFP 2.1 with other three types of searches. That is, we added the possibility to solve problems by using: *Depth-First Search* (DFS), *Iterative Depth-First Search* (I-DFS), and *Best-First Search*. Each of these searches is well-known among the planning community and, therefore, we will not provide any details on their implementation. As always Russell and Norvig [2010] propose an excellent review of the aforementioned topics. In Tables 5.15 to 5.19, we show some empirical evaluation of BFS, DFS, and I-DFS. As we can see from the results, none of the approaches is clearly better than the other and, depending on the domain we are trying to solve, one search strategy may be more advantageous than the others. Nevertheless, from our results, we can conclude that BFS has the best results in general and that I-DFS is almost always to prefer to DFS.

CB with $\|\mathcal{AG}\| = 3, \|\mathcal{F}\| = 8, \|\mathcal{A}\| = 21$			
L	BFS	I-DFS	DFS
2	**0.001**	0.004	0.186
3	**0.017**	0.022	0.879
5	0.249	**0.225**	13.894
6	2.287	**1.389**	139.884
7	6.233	**5.445**	452.234

Table 5.15: Solving times of the three uninformed searches of EFP 2.1 on the Coin in the Box (**CB**) domain.

AL with $\|\mathcal{AG}\| = 2, \|\mathcal{F}\| = 4, \|\mathcal{A}\| = 6$			
d	BFS	I-DFS	DFS
2	0.047	0.108	**0.016**
3	0.052	0.114	**0.016**
4	0.070	0.138	**0.019**
5	0.297	0.407	**0.031**
6	1.452	1.923	**0.094**
7	15.692	18.724	**0.384**
8	90.983	115.643	**1.693**
9	1003.423	1190.613	**7.638**
C	0.055	0.127	**0.019**

Table 5.16: Solving times of the three uninformed searches of EFP 2.1 on the Assembly Line (**AL**) domain.

SC with $\|\mathcal{AG}\| = 9, \|\mathcal{F}\| = 12, \|\mathcal{A}\| = 14$			
L	BFS	I-DFS	DFS
4	**0.006**	0.015	0.595
5	**0.013**	0.043	1.305
6	**0.031**	0.119	3.493
7	**0.085**	0.313	10.977
8	**0.235**	0.828	34.982
9	**0.061**	2.270	112.461
10	**1.604**	6.115	365.561
11	**4.513**	15.985	1190.163

Table 5.17: Solving times of the three uninformed searches of EFP 2.1 on the Selective Communication (**SC**) domain.

GR with $\|\mathcal{AG}\| = 3, \|\mathcal{F}\| = 9, \|\mathcal{A}\| = 24$				**GR** with $\|\mathcal{AG}\| = 4, \|\mathcal{F}\| = 12, \|\mathcal{A}\| = 42$			
L	BFS	I-DFS	DFS	L	BFS	I-DFS	DFS
2	**0.021**	0.045	1.125	2	0.105	**0.024**	4.328
3	0.201	**0.055**	6.664	3	0.760	**0.352**	35.544
4	0.871	**0.278**	45.544	4	5.248	**2.064**	324.338
5	5.3085	**2.534**	301.848	5	36.391	**19.153**	64.394
6	**14.784**	22.817	1001.633	6	**98.841**	253.634	211.478

Table 5.18: Solving times of the three uninformed searches of EFP 2.1 on the Grapevine (**GR**) domain.

CC with $\|\mathcal{AG}\| = 2, \|\mathcal{F}\| = 10, \|\mathcal{A}\| = 16$				**CC** with $\|\mathcal{AG}\| = 3, \|\mathcal{F}\| = 13, \|\mathcal{A}\| = 24$			
L	BFS	I-DFS	DFS	L	BFS	I-DFS	DFS
3	**0.016**	0.027	0.926	3	**0.023**	0.082	1.264
4	**0.026**	0.036	1.886	4	**0.0852**	0.208	6.289
5	0.179	**0.149**	15.106	5	**0.625**	1.765	0.029
6	0.629	**0.465**	76.496	6	**1.835**	1.957	290.614
7	2.625	0.995	414.385	7	16.813	**11.125**	74.776
8	5.312	6.338	1171.427				

Table 5.19: Solving times of the three uninformed searches of EFP 2.1 on the Collaboration and Communication (**CC**) domain.

Heuristics While BFS, DFS, and I-DFS are all *uninformed* searches—*i.e.*, they traverse the space using only information derived by the search-tree and not from the e-states themselves—Best-First Search selects, at each step, the *best* state, that is the one that is, supposedly, closer to the goal. The problem with this last approach lies in finding a good function to calculate the score of each e-state and, therefore, in understanding which e-state is the best one. These functions, known as *heuristics*, have been deeply studied in the planning community and are, nowadays, a standard concept. To avoid unnecessary clutter we will not discuss the theoretical basis of this concept addressing the interested reader to Russell and Norvig [2010], Keyder and Geffner [2008] for an exhaustive presentation of the topic.

As mentioned, the poor scalability of epistemic reasoners is an important issue. Being the community relatively new, it is normal that most of the research efforts are put into investigating the foundation of the problem rather than optimizing what already exists. Nonetheless, having tools that, most of the time, have not acceptable performances (with respect to classical planning, for example) limits the proliferation of the solvers themselves. It is paramount, in our opinion, to focus on the optimization of existing tools in order to have competitive epistemic reasoners that can be employed by other researchers or even in real-world scenarios. This would allow the community to gain more momentum and grow even faster. To better understand how heuristics may help in scaling the solving process we report, in Table 5.20, the comparison between the fastest configuration of EFP 2.1

while using BFS and the same configuration while exploiting the *perfect heuristic* (P-Heur). This heuristics represents the theoretical optimal we can hope to achieve and, therefore, provides an excellent example of the potential of Best-First search. P-Heur is assumed to always be able to derive the exact distance between an e-state and the goal in constant time. Since we (unfortunately) do not have access to such information, we emulated such behaviour by pre-computing the search-space beforehand—this operation is not accounted for in the solving time—and associating to each state its actual distance to the goal. While this is not really helpful in optimizing the search[2], it allows us to understand how well the solver could perform with the right heuristics.

CC with $\|\mathcal{AG}\| = 3, \mathcal{F}\| = 15, \|\mathcal{A}\| = 42$		
L	EFP 2.1	P-Heur
3	0.16	**0.06**
4	1.07	**0.07**
5	28.73	**0.09**
6	118.60	**0.13**
7	1427.35	**0.16**

Table 5.20: Comparison between the solving times of an uninformed search (EFP 2.1) and the theoretical optimal informed one (P-Heur) on the Collaboration and Communication (**CC**) domain.

That is why, as the final contribution to EFP 2.1, we decided to focus on formalizing some domain-independent heuristics for MEP. First of all, we implemented a module, called `heuristics_manager`, that allows the planner to make use of heuristics as black boxes. This allows any interested researcher to simply implement their heuristics and directly test it on EFP 2.1, without having to know in detail the solver structure. Moreover, we also formalized two diverse domain-independent heuristics for MEP problems: the *number of satisfied sub-goal* and an updated version of the *epistemic planning graph* presented in Le et al. [2018]. While these two heuristics are completely formalized, they are yet to be fully implemented.

[2]This process requires to explore the whole search-tree before even starting to plan.

The first is a very simple heuristic that simply associates an higher evaluation to e-states that satisfy more sub-goals. To better improve this heuristic we also defined functions that allows "to break" complex goals into a conjunction of simpler ones. That is, being each goal simply a belief formula that needs to be satisfied, we devised a way of producing more sub-goals from a single one to better distinguish between e-states.

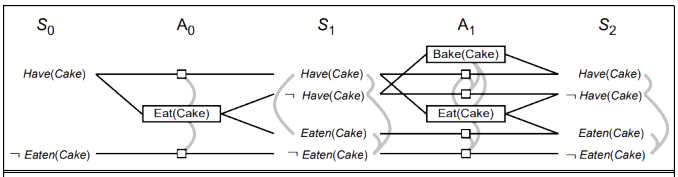
The second heuristic we envisioned is an updated version of the epistemic planning graph, that stems from a combination of the one presented in Le et al. [2018] and concepts derived from the ASP solver presented in the next section. This new planning graph is independent of the chosen e-state representation, making it available for every EFP 2.1 configuration (except for the one where custom update models are considered). While, as mentioned, we do not have a complete implementation of this feature yet, we can provide the theoretical details of its formalization.

First of all, let us quickly introduce the concept of *planning graph* in classical planning. A far more detailed and precise introduction to this topic can be found in Russell and Norvig [2010, chapter 10]. A planning graph is a special data structure used to generate heuristics using an algorithm called *GRAPHPLAN*. Intuitively a planning graph (Figure 5.2) is a directed graph organized into *levels*: first, a level S_0 for the initial state, consisting of nodes representing each fluent literal that holds in S_0; then a level A_0 consisting of all the ground actions that might be applicable in S_0; then, alternating, a level S_i followed by A_i; until we reach a termination condition. A more formal characterization of this structure is presented in Definition 5.1:

$Init(Have(Cake))$
$Goal(Have(Cake) \wedge Eaten(Cake))$
$Action(Eat(Cake)$
PRECOND: $Have(Cake)$
EFFECT: $\neg\, Have(Cake) \wedge Eaten(Cake))$
$Action(Bake(Cake)$
PRECOND: $\neg\, Have(Cake)$
EFFECT: $Have(Cake)$

The "have cake and eat cake too" problem.

(a) Problem description.



The planning graph for the "have cake and eat cake too" problem up to level S_2. Rectangles indicate actions (small squares indicate persistence actions) and straight lines indicate preconditions and effects. Mutex links are shown as curved gray lines.

(b) Planning graph of the problem in Figure 5.2a.

Figure 5.2: Example of a planning graph. Images extrapolated from Russell and Norvig [2010, chapter 10].

Definition 5.1: Planning Graph

Given a planning problem $P = \langle D, I, G\rangle$, the planning graph of P is an alternate sequence of state levels and action levels $S_0, A_0, \ldots, S_k, A_k, \ldots$ where

- S_0 represents I;
- for $i \geq 0$,
  - A_i is the set of actions executable in S_i; and
  - $S_{i+1} = S_i \cup \left(\bigcup_{a \in A_i} \Phi(S_i, a)\right)$. Where Φ is the transition function in P.

A planning graph gives important information about the problem in *polynomial* time. The idea is that, despite the possible errors, the level j at which a fluent literal first appears is a good estimate of how difficult it is to achieve that fluent literal from the initial state. Other important properties of the planning graph are that: the estimation is always correct when it reports that the goal is not reachable; and that it never overestimates the number of steps, generating an admissible heuristic. The termination of the construction of the planning graph (when the search space is finite) is also guaranteed through saturation. While the state levels are "easily"

defined in the classical settings, the same is not true in the epistemic scenario. In fact, simply putting all the fluent literals in the states would not capture enough information, and using complete e-states would result in a massive overhead, given that they are graph-like structures and their manipulation is very resource-heavy.

The *epistemic planning graph* (ePG 1.0) presented in Le et al. [2018] solved this problem by defining each state level as a set of partial Kripke structures. This allowed capturing enough information without aggravating the ePG 1.0 resource consumption. Nonetheless, this choice did not allow ePG 1.0 to work whenever a goal with negated beliefs was requested by the problem. To overcome such problem we decided to re-design the state level in a new version of the planning graph, called ePG 2.0. In particular, we envisioned a state level that is comprised of a set of instantiated belief formulae associated with a Boolean value. These formulae are all the ones that appear in the domain description, *i.e.*, single fluent literals, initial descriptions, goals, actions preconditions, actions effects, observability conditions, and so on. More practically, we envisioned the state level to be formed by two maps $\mathcal{P}$ and $\mathcal{Q}$. The former associates the extrapolated belief formulae to either True or False, while the latter associate each fluent literal to a Boolean value.

Before providing more information on how ePG 2.0 may be used to compute e-states score we need to present how the entailment (to check for action executability and other conditions) and the execution of an action work in this structure. Let us start by giving the definition of entailment.

Definition 5.2: ePG 2.0 Entailment

Given a state level S_i, its relative maps $\mathcal{P}_i$ and $\mathcal{Q}_i$ and a belief formula φ we have that $S_i \mid\!\sim \varphi$ (where $\mid\!\sim$ indicates the entailment in ePG 2.0) if:

- φ is fluent literal or its negation: $\mathcal{Q}_i$ associates φ to $\top$; or
- φ is a belief formula: $\mathcal{P}_i$ associates φ to $\top$.

Since each fluent and each formula in ePG 2.0 is associated with a Boolean value, we have that the entailment is simply derived by reading such values. This means that whenever an information is associated to true in state level, then that information

is entailed by the level. On the other hand, if a fluent or a formula is associated to false in a level, then it is not entailed in that level.

We can now explain how an action execution works in ePG 2.0. Given an executable[3] action `a` and a state level S_i we have that `a` could potentially set to true the Boolean value associated with any fluent literal[4] or belief formula. It is important to notice that once any entry of the two maps is set to True, it will always maintain this status. This behavior emulates the insertion of fluent literals in the states level in the classical version of the planning graph. Intuitively, if the effects of an action consider a fluent literal `f` then the update will check if this fluent, its negation, or the belief formulae that have this `f` as part of their argument, can be set to True. To check whether a fluent literal (or its negation) is verified after the execution of an action it is not too intricate. In fact, a fluent can only be manipulated by ontic actions which clearly state the new value of the fluent literal they consider. That is, if an ontic action makes the fluent literal `f` True than $\mathcal{Q}[\texttt{f}] = \top$, otherwise, if the action sets `f` to be False, we will have $\mathcal{Q}[\neg\texttt{f}] = \top$. A more precise definition of this procedure is illustrated in Algorithm 2.

Algorithm 2: Fluent value updater

```
   Input  : Q   //The map ⟨fluent, bool⟩ of S_i
            a   //The action executed on S_i
   Output : Q   //The updated version of the input map
 1 //Note that ℓ may also be a negated fluent
 2 for fluent ℓ in Q.get_keys() do
 3     if Q[ℓ].get_value() == ⊥ then
 4         //An effect is considered if its condition are True
 5         if a.get_effects().contains(ℓ) then
 6             Q[ℓ].set_value(⊤)
 7         end
 8     end
 9 end
10 return Q
```

[3] An action is executable in a state level S_i if its executability conditions are entailed by S_i.

[4] Let us remember that each fluent literal and its negation are independent and are considered as separate entries in $\mathcal{Q}$.

Contrarily to the fluent literals check, determining whether a belief formula is verified after an action execution is not straightforward. The first complication resides in the fact that an action verifies an infinite number of belief formulae considering all the possible beliefs chains. That is why, we need to check only the belief formulae of interest, *i.e.*, the ones contained in $\mathcal{P}$. For the sake of the readability we will not attempt to describe how this check works using plain text, instead, we will make use of a much more concise pseudo-code. Algorithm 3 is the function that manages this update. In particular, Algorithm 4 presents this procedure for ontic actions while Algorithm 5 shows the one for sensing and announcement actions.

Algorithm 3: Check belief formula after action execution

```
   Input  : P   //The map ⟨belief_formula, bool⟩ of S_i
            a   //The epistemic action executed on S_i
   Output : P   //The updated map ⟨belief_formula, bool⟩ of S_i
 1 for belief_formula bf in P.get_keys() do
 2     if P[bf].get_value() == ⊥ then
 3         if P[bf].get_type() == ontic then
 4             //Call to Algorithm 4
 5             bool res = check_bf(bf, a, P)
 6         else
 7             //Call to Algorithm 5
 8             bool res = check_bf_epi(bf, a, P, 0)
 9         end
10     end
11     P[bf].set_value(res)
12 end
13 return P
```

Algorithm 4: Check belief formula after ontic execution

```
   Input  : bf      //The belief formula to verify
            P       //The map ⟨belief_formula, bool⟩ of S_i
            a       //The ontic action executed on S_i
   Output : ⊤ or ⊥  //Depending on the updated value of P[bf]
 1 if P[bf].get_value() == ⊤ then
 2 |  return ⊤
 3 end
 4
 5 if bf.get_base_fluents().contains_one(a.get_effects()) then
 6 |  if bf.get_type() == fluent_formula then
 7 |  |  //For simplicity, assume fluent formulae of one fluent
 8 |  |  P[bf].set_value(⊤)
 9 |  |  return ⊤
10 |  else if bf.get_type() == single_ag_belief then
11 |  |  if fully_obs.contains(bf.get_agent()) then
12 |  |  |  //Recursive call to this function
13 |  |  |  return check_bf(bf.get_nested_bf(), a, P)
14 |  |  end
15 |  else if bf.get_type() == group_formula then
16 |  |  if fully_obs.contains(bf.get_group_agents()) then
17 |  |  |  //Recursive call to this function
18 |  |  |  return check_bf(bf.get_nested_bf(), a, P)
19 |  |  end
20 |  else
21 |  |  /*Disjunction and conjunction of belief formulae follow
   |  |   the standard semantics of these operators*/
22 |  end
23 end
24 return ⊥
```

Algorithm 5: Check belief formula after epistemic execution

```
Input  : bf      //The belief formula to verify
         P       //The map ⟨belief_formula, bool⟩ of S_i
         a       //The ontic action executed on S_i
         x       //Label used to differentiate scenarios
Output : ⊤ or ⊥  //Depending on the updated value of P[bf]
1  if P[bf].get_value() == ⊤ then
2  |  return ⊤
3  end
4
5  //We check also for negated effects for the Partial observers
6  if bf.get_base_fluent().contains_one_or_negated(a.get_effects()) then
7  |  if bf.get_type() == fluent_formula then
8  |  |  if a.get_effects().contains(bf.get_base_fluent()) and x == 0 then
9  |  |  |  return ⊤
10 |  |  else if x == 1 then
11 |  |  |  return ⊤
12 |  |  else
13 |  |  |  return ⊥
14 |  |  end
15 |  else if bf.get_type() == single_ag_belief then
16 |  |  if fully_obs.contains(bf.get_agent()) then
17 |  |  |  if x == 2 then
18 |  |  |  |  x = 1
19 |  |  |  end
20 |  |  |  return check_bf(bf.get_nested_bf(), a, P, x)
21 |  |  else if partially_obs.contains(bf.get_agent()) then
22 |  |  |  return check_bf(bf.get_nested_bf(), a, P, 2)
23 |  else if bf.get_type() == group_formula then
24 |  |  if fully_obs.contains(bf.get_group_agents()) then
25 |  |  |  return check_bf(bf.get_nested_bf(), a, P, 0)
26 |  |  else if partially_obs.contains_one(bf.get_group_agents()) then
27 |  |  |  return check_bf(bf.get_nested_bf(), a, P, 2)
28 |  else
29 |  |  /*Disjunction and conjunction of belief formulae follow
   |  |   the standard semantics of these operators*/
30 |  end
31 end
32 return ⊥
```

To summarize, the construction of the planning graph is comprised of the following steps:

- First of all we build the initial state level, *i.e.*, S_0, where the maps $\mathcal{P}_0$ and $\mathcal{Q}_0$ will associate all the belief formulae of interest (the ones found in the domain description) and all the fluent literals (also negated) to the Boolean value False.
- We, then, check the conditions that are used to generate the initial state and set to true all the beliefs formulae and fluent literals that are verified in this e-state.
- After that, we iteratively execute the following procedure until the goal is satisfied by one of the state levels or we reach a fixed point[5]:
  - We check if the state level entails all the goal conditions. If it does, we found the goal.
  - If the goal is not found we then execute all the executable actions on the state level producing a new one.
  - If the new state differs, *i.e.*, has some new verified fluent literals or belief formulae, we reiterate the procedure, otherwise we reached the fixed point and we conclude that the problem cannot be solved.

Finally, once the planning graph has been built, we can extrapolate useful information following standard approaches presented in Le et al. [2018].

[5]A fixed point is reached whenever a state level and its successor are identical.

5.3 PLATO: an Epistemic Planner in ASP

In this section, following the idea originally proposed in 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. In particular, we present an actual implementation of a multi-shot ASP-based planner, called PLATO (ePistemic muLti-agent Answer seT programming sOlver), that can reason on domains described using $m\mathcal{A}^{\rho}$. 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 ASP paradigm enables a concise and elegant design of the planner, with respect to other imperative implementations, facilitating the development of formal verification of correctness. In particular, the declarative encoding allows us to provide formal proofs of results correctness, which are presented later in this chapter. Moreover, the planner, exploiting an ad-hoc epistemic state representation and the efficiency of ASP solvers, maintains competitive performance results on benchmarks collected from the literature.

5.3.1 Modeling MEP using ASP

Let us now present the details of the multi-shot ASP encoding for a multi-agent epistemic planning domain $D = \langle \mathcal{F}, \mathcal{AG}, \mathcal{A}, \varphi_{ini}, \varphi_{goal} \rangle$ (Definition 1.15) upon the possibilities based semantics described in Section 2.2. 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* by Gebser et al. [2019].

Epistemic states

Let us start by defining how an e-state, and specifically a possibility, is defined in PLATO. To do that, following Definition 2.9, we need to encode the possible worlds and the agents' beliefs. We use atoms of the form `pos_w(T, R, P)` and `believes(T1, R1, P1, T2, R2, P2, AG)`, respectively. Intuitively, the first atom

identifies a possibility with the triple (T, R, P), while the second encodes an "edge" between the possibilities (T1, R1, P1) and (T2, R2, P2), labeled with the agent AG.

Let us now focus in more detail on `pos_w(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. Afterward, 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 `i` be an agent and `u` and `v` be two possibilities represented by the triples (Tu, Ru, Pu) and (Tv, Rv, Pv), respectively. Then, we encode the fact that $\mathtt{v} \in \mathtt{u}(\mathtt{i})$ with the atom `believes(Tu, Ru, Pu, Tv, Rv, Pv, i)`.

The truth value of each fluent is captured by an atom of the form `holds(Tu, Ru, Pu, F)`. The truth of this 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 (Tu, Ru, Pu) when this will cause no ambiguity.

Entailment

To verify if a given belief formula (Definition 1.10) `F` is entailed by a possibility, we use the predicate `entails(P, F)` that follows Definition 2.11, 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).
```

Initial state generation

As mentioned above, following Son et al. [2014], we assume the initial state to model a finitary **S5**-theory. This means that the formulae that shape the initial state have a constrained structure. While detailed descriptions of such formulae are explored in Son et al. [2014], let us only provide a high-level characterization of these formulae for the sake of simplicity. Let ψ be a fluent formula, $\mathtt{f} \in D(\mathcal{F})$ be a fluent, $\mathtt{i} \in D(\mathcal{AG})$ be an agent, and let us use $\mathcal{AG}$ instead of $D(\mathcal{AG})$ for the sake of readability. Consider a $m\mathcal{A}^\rho$ statement of the form [**initially** φ] $\in D$; we have five cases:

(i) $\varphi \equiv \mathtt{f}/\neg\mathtt{f}$: `f` must (not) hold in the pointed possibility.

(ii) $\varphi \equiv \mathbf{C}_{\mathcal{AG}}(\mathtt{f}/\neg\mathtt{f})$: `f` must (not) hold in each possibility of the initial state.

(iii) $\varphi \equiv \mathbf{C}_{\mathcal{AG}}(\psi)$: if ψ is a fluent formula that is *not* a fluent literal, then it must be entailed from each possibility of the initial state.

(iv) $\varphi \equiv \mathbf{C}_{\mathcal{AG}}(\mathbf{B}_{\mathsf{i}}(\psi) \vee \mathbf{B}_{\mathsf{i}}(\neg\psi))$: 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 believes whether ψ is true in the initial state.

(v) $\varphi \equiv \mathbf{C}_{\mathcal{AG}}(\neg\mathbf{B}_{\mathsf{i}}(\psi) \wedge \neg\mathbf{B}_{\mathsf{i}}(\neg\psi))$: this type of formula expresses the fact that agent i does *not* believe 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 *(i)– (iii)* 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** $\mathbf{C}_{\mathcal{AG}}(\mathtt{f})$] or [**initially** $\mathbf{C}_{\mathcal{AG}}(\neg\mathtt{f})$]. In the former case, all agents will believe 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)`[6], 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 *(iii)*. We create a possible world `pos_w(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 *(i)* formulae. Finally, formulae of type *(iv)* 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 *(iv)* formulae ψ such that `P1` and `P2` do *not* agree

[6]Let us note that we use `0` to indicate the initial state *time* parameter.

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, $\forall \mathsf{u} \in \mathcal{G}, \forall \mathsf{i} \in \mathcal{AG}$ we have that $\mathsf{u}(\mathsf{i}) = \mathcal{G}$. We can observe that type *(v)* 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 $\mathtt{T} > 0$. $m\mathcal{A}^{\rho}$ makes use of three distinct types of actions—ontic, sensing, and announcement (Definition 2.12)—but for all of them the implementation of executability conditions is the same. 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, 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 $(\mathtt{T2}, \mathtt{R2}, \mathtt{P2})$. 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.

Ontic actions Let `ACT` be an ontic action executed at time `T` and let $\mathsf{u} = (\mathtt{T\text{-}1}, \mathtt{RP}, \mathtt{PP})$ 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 $\mathsf{w} \in \mathsf{u}(\mathsf{i})$, for each fully observant agent `i`; 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 `pos_w` (short for `possible_world`) is defined as follows:

```
pos_w  (T, R2 + MR + 1, P2):-
       pointed(T-1, RP, PP), pos_w(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 $\mathtt{F_{ACT}}$ 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 with the pointed possibility itself. In this way, we can 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 2.1, 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 `pos_w` atoms.

Sensing/Announcement actions As shown in Definition 2.12 behavior of sensing and announcement actions are similar. 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:

(i) `P2` = `PP` (here we also set `P2`$'$ as the pointed possibility at time `T`);

(ii) `P2` is reached by a fully observant path and it is consistent with the effects of `ACT`;

(iii) `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 to be consistent with the changes, after an action is performed (that is, we do not want to carry old information obtained in previous states). Similar to ontic actions, condition *(ii)* deals with the possibilities believed by fully observant agents; if `i` is fully observant, then she must only believe those possible worlds that are consistent with the effects of `ACT`. Finally, condition *(iii)* 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 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 fully observant agents is entirely captured by condition *(ii)*.

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 with respect to the beliefs of the fully observant ones. We deal with oblivious agents exactly as for ontic actions.

Optimizations

To minimize the amount of ground `pos_w` atoms, we designed the function so that it reuses, 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 with respect to their fluents. Specifically, let `P1` and `P2` be two initial possibilities. Let `MFi = #max { F : holds(Pi, F), not holds(Pj, F) }`, with $\texttt{i} \neq \texttt{j}$. Then we impose that if $\texttt{P1} < \texttt{P2}$, then it must also hold that $\texttt{MF1} < \texttt{MF2}$. 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. [2019] we divided our ASP program into three main sub-programs, where the parameter `t` stands for the execution time of the actions: (1) `base`: it contains the rules for the generation of the initial state ($\texttt{t} = 0$), alongside with the instance encoding; (2) `step(t)`: it deals with the plan generation ($\texttt{t} > 0$) and with the application of the transition function; and (3) `check(t)`: it verifies whether the goal is reached at time $\texttt{t} \geq 0$.

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 $\texttt{t} + 1$ (method `release_external`). Using the Python script provided by Gebser et al. [2019], 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 ground and solved; we check the goal constraint until the condition is satisfied.

5.3.2 Experimental Evaluation

In this Section we compare PLATO to the multi-agent epistemic planner EFP 2.1. 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 $\mathbf{B}$ in the DEL formulae;
- K-BIS/P-MAR: the solver EFP 2.1 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 by Dovier [2015]).

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 similar behaviors, as shown in Table 5.21a, 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.

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 (**SC**).

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 (**GR**).

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 (**CB**).

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 (**AL**).

	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 (**CC**).

Table 5.21: (a) Comparison of `frumpy`, `many` and EFP 2.1 on **SC**. **(b)** Total, grounding and solving times for **GR** using `multi`. The last column reports the number of ground atoms. **(c)** Comparison of `multi` and `bis` on **CB**. **(d)** Comparison of PLATO and EFP 2.1 on **AL** (**C** identifies that the executability conditions are expressed through common beliefs). **(e)** Comparison of `single`, `multi` and EFP 2.1 on **CC**.

To evaluate the behavior of PLATO with respect to the entailment of DEL formulae, we exploited the **AL** domain (Table 5.21d), 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 beliefs in the executability conditions leads to worse results. This is because 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 from its depth, is handled by a ground atom and, therefore, a higher depth does not impact the solving process. On the other hand, the entailment in EFP 2.1 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 5.21b 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 5.21c), due to a much heavier contribution of the grounding phase.

Finally, we compare the single-shot/multi-shot encodings in Table 5.21e. 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. [2019], this is due to the reuse of *nogoods* learned from previous solving steps.

5.3.3 Correctness of PLATO

Declarative languages such as ASP allow a high-level implementation, facilitating the derivation of formal verification of correctness. Considering 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 Definition 1.15). Finally, we express the entailment with respect to $m\mathcal{A}^{\rho}$ and PLATO with $\models_{\Phi}$ and $\models_{\Gamma}$, respectively. Each main component of the planner is addressed by one proposition among Propositions 5.1 to 5.3. Proofs of these properties are reported in Appendix A.5.

Proposition 5.1: PLATO Entailment Correctness

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 5.2: PLATO Initial State Construction Correctness

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 5.3: PLATO Transition Function Correctness

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* $\Gamma(\mathsf{a}, \mathsf{v}) \models_\Gamma \psi$.

These results allowed us to verify the empirical correctness of the planner EFP 2.1. In all of the conducted tests, the two planners exhibited the same behavior. 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 2.1 and PLATO are unable to determine if a solution for a planning problem exists (even when checking for already visited states).

Nothing in life is as important as you think it is when you are thinking about it.

— Daniel Kahneman
Thinking, Fast and Slow
[Kahneman, 2011]

6

"Fast and Slow" Epistemic Planning

Contents

6.1 Background

Artificial Intelligence-based systems have been the focal point of computer science research in the last years. This led to the creation of several automated tools and successful applications that are pervading our everyday life. Nonetheless, most of these systems can be considered instances of *narrow AI*: *i.e.*, they are, generally, focused on a limited set of abilities and goals. Ultimately, these approaches are becoming more and more efficient in dealing with their pre-established areas of interest thanks to improved algorithms and techniques, and also, especially in the case of *Machine Learning* (ML) systems, thanks to the availability of huge datasets

and computational power [Marcus, 2020]. On the other hand, all of these tools still lack many capabilities that, we humans, naturally consider to be included in a notion of "intelligence". Examples of these capabilities are generalizability, robustness, explainability, causal analysis, abstraction, common sense reasoning, ethical reasoning, as well as a complex and seamless integration of learning and reasoning supported by both implicit and explicit knowledge. That is why the majority of the AI community is attempting to address these current limitations and it is trying to create systems that display more "human-like qualities". One of the central debates is whether end-to-end *neural networks* or *symbolic* and *logic-based AI* approaches alone can achieve this goal or whether we need to integrate these techniques to achieve the desired AI system.

We believe the integration route to be the most promising. This idea is also supported by several results that have been obtained along this line of work. For example, Marcus [2020] argues that symbolic and logic-based reasoning is paramount to improve the robustness of AI systems. As pointed out in Besold et al. [2017], Kotseruba and Tsotsos [2020], several research groups are building "hybrid" approaches that use both machine learning and symbolic reasoning techniques, employing a so-called neuro-symbolic AI approach.

We argue that a better comprehension of how humans have, and have evolved to obtain, these advanced capabilities can inspire innovative ways to imbue Artificial Intelligence systems with these competencies. More precisely, we analyzed some of these theories, with a special focus on the theory of *thinking fast and slow* presented by Kahneman [2011], and attempted to translate them into an AI environment, conjecturing that this will lead to an advancement in machines capabilities. This chapter gives a brief and high-level overview of this general approach, providing also an early implementation of a "hybrid" AI architecture that focuses on the MEP setting.

6.1.1 Theories of Human Decision Making

According to the book "*Thinking, Fast and Slow*" by Kahneman [2011], humans' decision-making processes are guided by the cooperation of two capabilities, that, are referred to as "`System`s". In particular, `System-1` provides tools for intuitive, imprecise, fast, and often unconscious decisions ("thinking fast"), while `System-2` handles more complex situations where logical and rational thinking is needed to reach a complex decision ("thinking slow"). The former is guided mainly by intuition and experience rather than deliberation and allows to quickly formulate answers to very simple questions. Such answers may be sometimes wrong, mainly because of unconscious biases or because they rely on shortcuts, and usually come with no explanation. However, `System-1` is able to build models of the world that, although inaccurate and imprecise, can fill the knowledge gaps through causal inference and allow us to respond reasonably well to the many stimuli of our everyday life. A typical example of a task handled by `System-1` is finding the answer to a very simple arithmetic calculation, or reaching out to grab something that is going to fall. We use our `System-1` about 95% of the time when we need to make a decision. On the other hand, whenever the problems to be solved starts to become too demanding, `System-2`, thanks to the access to additional "computational resources" and rational/logical thinking, is the one that is in charge of their resolution. A typical example of a problem handled by `System-2` is solving a complex arithmetic calculation, or a multi-criteria optimization problem. To do this, humans need to be able to recognize that a problem goes beyond a threshold of cognitive ease and therefore they need to activate a more global and accurate reasoning machinery. Hence, introspection is essential in this process.

Other than the idea of problem difficulty `System-1` and `System-2` discern which problem they should tackle based on the experience accumulated on the problem itself. That is, when a *new* non-trivial problem has to be solved, it is handled by `System-2`. However, certain problems over time, and therefore after having accumulated a certain amount of experience, pass on to `System-1`. The reason is that the procedures used by `System-2` to find solutions to such problems also

accumulate examples that `System-1` can later use readily with little effort. A typical example is reading text in our native language. However, this does not happen with all tasks, *e.g.*, finding the correct solution to complex arithmetic questions. Finally, Kahneman theorizes that `System-2` may employ heuristics to facilitate the exploration of the search space, especially when this is very large. These heuristics could derive from `System-1` and usually help in focusing the attention only on the most promising parts of the space, allowing `System-2` to work with manageable time and space. Thanks to this "structure", humans are able to consider diverse levels of abstraction, adapt, and generalize their experiences while also being able to multi-task when using their `System-1`. We envisioned `System-2` to be sequential, given that it requires full attention, limiting the number of complex problems that can be solved by humans in parallel to one. Let us note, however, that `System-1` and `System-2` are not systems in the multi-agent sense, but rather they encapsulate two wide classes of information processing.

Kahneman's theory gives a detailed account on how we make decisions, while others conjecture what are the reasons behind the evolution of our reasoning scheme—*e.g.*, Harari [2015] identifies the ability to conceive and communicate high-level stories as one of the main reasons. Nevertheless, in most of these theories it is clear that the notions of *consciousness* and *abstraction* are important to identify the traits of intelligence. These provide the ability to consciously focus attention on a limited set of features, while deferring others, to process a specific task in depth. Graziano [2013], Graziano et al. [2020] envisioned two forms of consciousness in human beings: the **I**-consciousness (**I** for Information) and the **M**-consciousness (**M** for mysterious). The first one refers to the ability to solve (possibly complex) problems, by recognizing necessary processing steps in specific (even new) contexts, to tackle a desired problem. Again, these concepts seem to intertwine with Kahneman's theory. The former could be seen as another way to identify `System-2` since it has to do with considering a problem and harnessing the relevant faculties of our cognition to devise a plan to solve it. The latter refers to our ability to build a simplified, approximate, and subjective model of peoples', both ourselves and others, minds,

beliefs, and intentions. Such low-fidelity model building can be linked to `System-1`, as `System-1` is able to form a rapid but usually inexact model of the world.

6.1.2 AI Thinking, Fast and Slow

The theories described in the previous paragraph, as well as their connections, shed some light on which competencies provide humans the ability to solve a diverse set of simple and complex problems; understand broad contexts robustly; adapt readily; and ultimately cooperate. These competencies are, arguably, what makes our intelligence *broad*, in opposition to the narrow one displayed by modern AI systems. This difference makes arise several interesting research questions about the capabilities that AI systems should include in the future. The "Blue Sky" paper by Booch et al. [2021] reports some of these questions. In this chapter, we will focus on the first and part of the fifth research questions posed by Booch et al.. Namely:

1. "*Should we clearly identify the AI* `System-1` *and* `System-2` *capabilities? What would their features be? Should there be two sets of capabilities or more?*"

5. "*How do we model the governance of* `System-1` *and* `System-2` *in an AI? When do we switch or combine them? Which factors trigger the switch?* [...]"

While the authors of the Blue Sky paper did not define these research objectives for any particular AI system, in this chapter we will try to address them in the Multi-agent Epistemic Planning setting.

6.2 MEP `System-1` and `System-2`

Two of the prominent lines of work in AI, *i.e.*, machine learning and symbolic reasoning, seem to embody (even if loosely) the two `System`s presented above. In particular, ML is a data-driven approach to AI and shares with `System-1` its ability to build (possibly imprecise and biased) models from sensory data. Perception activities, such as seeing, that in humans are handled by `System-1`, are currently addressed with machine learning techniques in AI. However, some traits of `System-1` do not seem to be present, at least for now, in ML. Examples of these are the ability to

grasp basic notions of causality and common-sense reasoning. Similarly, System-2's capability to solve complex problems using a knowledge-based approach is somewhat emulated by AI techniques based on logic, search, and planning, that make use of explicit and well-structured knowledge. While the parallelism ML–System-1 and logic programming–System-2 represents a starting point in developing an automated fast and slow AI, we should not assume these two techniques to be the exclusive representative of the respective System.

In what follows, we will try to give a characterization of both a System-1 and a System-2 transposition to automated tools, referred to as *solvers* for brevity. We will start with general definitions of such solvers only to present, later in the chapter, actual implementations of System-1 and System-2 reasoners in the epistemic setting. We will make use of three *models* to represent key modules of our abstract **Reasoner**[1]. In particular, the *model of self* is used to store the experience of the architecture, the *model of the world* contains the knowledge accumulated by the system over the external environment and the expected tasks, while the *model of others* contains the knowledge and beliefs about other agents who may act in the same environment. Finally, the *model updater* acts in the background to keep all models updated as new knowledge of the world, of other agents, or new decisions are generated and evaluated.

The general characterization of a System-1 solver, triggered immediately when the problem is presented to the **Reasoner**, does not require many factors.

- These solvers are assumed to rely on the past experience of the **Reasoner** itself.
- Moreover, we assume that the running time for System-1 approaches to be independent of the input and, instead, to depend on the experience accumulated by the overall architecture, in the *model of self*.

[1]We will use this term to indicate an abstract entity that acts as a proxy for an architecture that contains and manages various System-1 and System-2 solvers. Let us imagine the **Reasoner** to be an artificial version of the human body which has its low-level reasoning capabilities defined by various System-1 and System-2.

- Finally, we consider a `System-1` solver to be an entity that relies on "intuition" (with a slight abuse of notation).

Considering these characteristics, the next question that naturally arises is *can MEP ever be considered as a `System-1` task, considering that epistemic planners, in literature, always rely on look-ahead strategies?* We considered some ideas that could help us develop a `System-1` epistemic planner. Among those, only a few were not using search methods (intensively) but rather mostly relied on experience. Finally, we identified a feasible, yet functional, way to exploit experience in the epistemic planning setting. The idea is to make use of *pre-computed plans*; that is, `System-1` can be used to determine which of the plans already generated by past experiences is the one that "fits the best" the current problem. Of course, determining if an already computed plan is a good choice or not for the current problem is a difficult research question on its own. Since the focus of this last chapter is to devise an overall fast and slow architecture for epistemic planning rather than optimizing its internal components, we decided to use a very simple criterion to select the best fitting plan. In particular, `System-1` selects, among past solutions for the same domain, the pre-computed plan that satisfies the most number of sub-goals of the problem that is being tackled. Let us remark that this is just an early-stage idea that could certainly be enriched and optimized.

Our **`Reasoner`** is a `System-1`-by-default architecture: whenever a new problem is presented, a `System-1` solver with the necessary skills to solve the problem starts working on it, generating a solution and a confidence level. This allows to minimize the resource consumption making use of the much faster `System-1` solving process when there is no need for `System-2`—that is when the solution proposed by `System-1` is "good enough". Nevertheless, as for the human brain, `System-1` may encounter problems that it cannot solve, either due to its lack of experience or the inherent intricacy of the problem itself. These situations require, then, the use of more thought-out resolution processes, generally provided by `System-2` approaches. Notice that we do not assume `System-2` solvers to be always better than `System-1` solvers: given enough experience, some tasks could be better solved

by System-1 solvers. This behavior also happens in human reasoning [Gigerenzer and Brighton, 2009]. In the particular case of MEP, we can consider as System-2 solving procedures the tools that employ traditional planning strategies. These can be, for example, the planner RP-MEP presented by Muise et al. [2015] and EFP 2.1 presented in Chapter 5. While these two solvers adopt different strategies to solve a Multi-agent Epistemic Planning problem, they both explore the search space and do not rely on experience.

6.2.1 Meta-cognition

One of the research questions posed by Booch et al. [2021] asks how "*do we model the governance of System-1 and System-2 in an AI?*". To address this we decided to focus on the idea of meta-cognition as firstly defined by Flavell [1979], Nelson [1990]. This means that we want our **Reasoner** to be equipped with a set of mechanisms that would allow it to both monitor and control its own cognitive activities, processes, and structures. The goal of this form of control is to improve the quality of the system's decisions [Cox and Raja, 2011]. Meta-cognition models have been largely studied [Cox, 2005, Kralik et al., 2018, Kotseruba and Tsotsos, 2020, Posner, 2020] in the past years. Among the various proposed modalities, we envisioned our **Reasoner** to have a centralized meta-cognitive module that exploits both internal and external data and arbitrates between System-1 and System-2 solvers. Let us note that this module is structurally and conceptually different from an algorithm portfolio selection [Kerschke et al., 2019, Tarzariol, 2019].

We propose a meta-cognitive (**MC**) module that itself follows the *thinking fast and slow* paradigm. This means that our **MC** module is comprised of two main phases: the first one takes intuitive decisions without considering many factors, while the second one is in charge of carefully selecting the best solving strategy, considering all the available elements whenever the first phase did not manage to return an adequate solution. We will refer to the former with **MC-1** and to the latter with **MC-2**. **MC-1** defines the System-1 part of the metacognitive process and, therefore, it activates automatically as a new task arrives. This module is

in charge of deciding whether to accept the solution proposed by the `System-1` solver or to activate **MC-2**. **MC-1** takes this decision considering the *confidence* of the `System-1` solver: if the confidence, which usually depends on the amount of experience, is high enough, **MC-1** adopts the `System-1` solver's solution.

If **MC-1** decides that the solution of the `System-1` solver is not "good enough", it engages **MC-2**. Intuitively, this module needs to evaluate whether to accept the solution proposed by the `System-1` solver or which `System-2` solver to activate for the task. To do this, **MC-2** compares the expected reward for the `System-2` solver with the expected reward of the `System-1` one: if the expected additional reward of running the `System-2` solver, compared to the `System-1` one, is large enough, then **MC-2** activates the `System-2` solver. **MC-2**, following the human reasoning model [Shenhav et al., 2013], is designed to avoid costly reasoning processes unless the additional cost is compensated by an even greater expected reward for the solution that the `System-2` solver will devise.

In what follows, we try to provide a "concrete" view of the `System-1`/`System-2` framework for the Multi-agent Epistemic Planning setting. The **MC-1** schema does not require a graphical visualization given that it only has to execute one decision. The same is not true for **MC-2**. That is why we will present a schematic view of this module in Figure 6.1. In the schema we will make use of the following notations for the sake of readability:

- **Planner-1** and **Planner-2** indicate the planners RP-MEP [Muise et al., 2015] and EFP 2.1, respectively.
- **Sys1** represents the solution obtained by the `System-1` solver.
- The variables d and *limits* are given as input.
- R(`S`) represents a formula that computes the reward of choosing the solver `S`. This formula is fully characterized by Ganapini et al. [2022]. Since we want our schema to be at an intuitive level we will not provide further details.

- Finally, the methods to simplify the problems are, at the moment, to restrict the belief formulae depth or to eliminate some sub-goals.

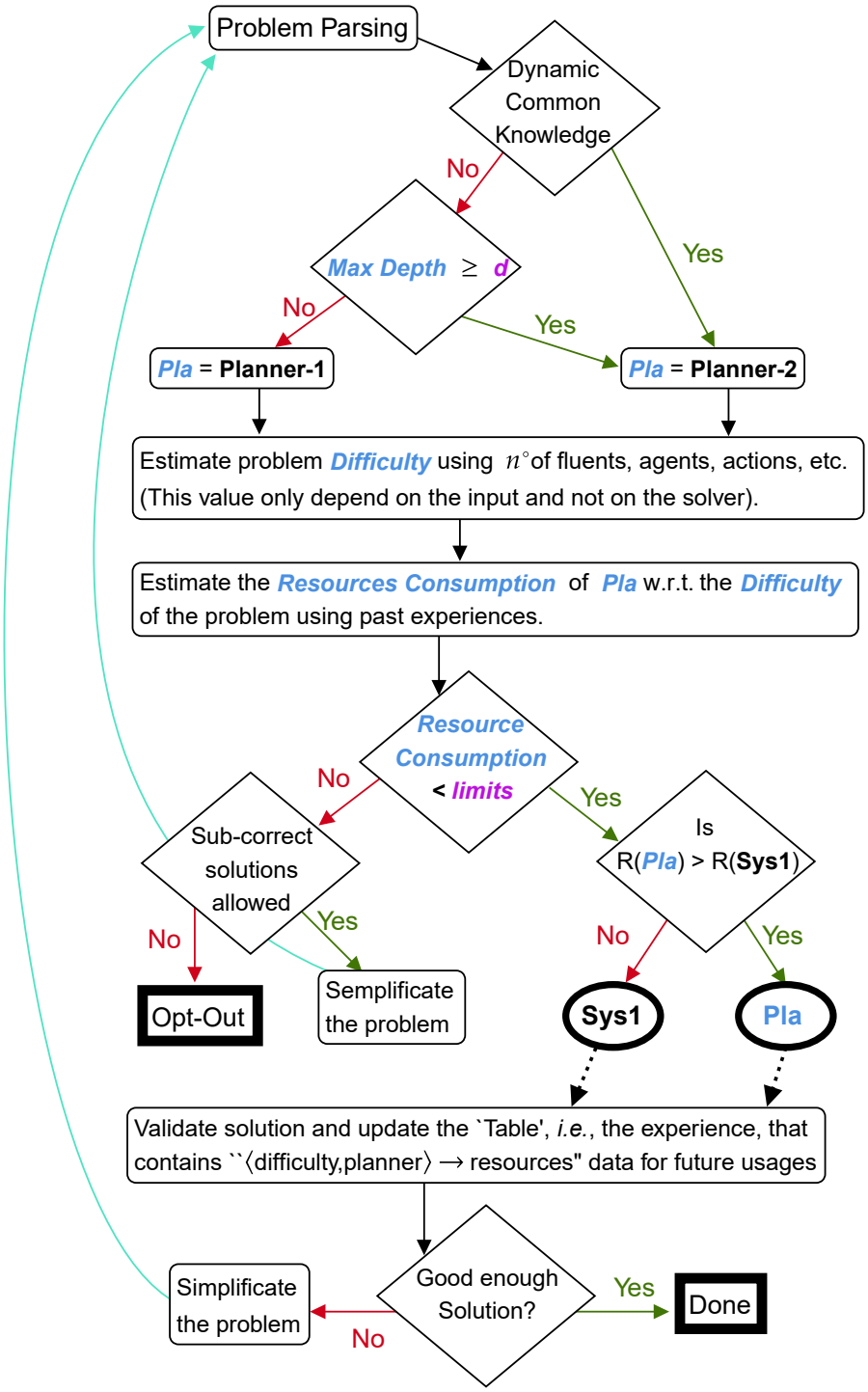


Figure 6.1: The schema of **MC-2**.

6.3 A Fast and Slow Epistemic Architecture

In this section, we will provide a description of an early implementation of the architecture described before. This tool can be found at `https://github.com/FrancescoFabiano/MetacognitiveEpistemicPlanning` and, while tries to emulate the behavior discussed in the previous sections, it also adopts some simplifications given that is still a premature version. Nonetheless, this tool is able to accomplish the basic functionalities and we believe it to be an excellent starting point to analyze the thinking fast and slow paradigm in MEP. Moreover, we envisioned this architecture to be easily modified by the other research groups, which can easily inject new (`System-1`/`System-2`) solvers and modify the **MC** modules to create a well-structured epistemic reasoner that will benefit from all the community research efforts.

6.3.1 E-PDDL: Standardized MEP Problems Language

As the first step in implementing our architecture, we addressed the problem of having a unified specification language. In fact, it is necessary that problems can be "understood" by all the solvers that could potentially tackle them. Over the years, the MEP community has developed multiple approaches with varying behaviors and specification languages. And while the diversity of approaches has led to a deeper understanding of the problem space, the community now lacks a standardized way to specify MEP problems. To address the situation, we propose a unified input description language for the epistemic setting, namely the *Epistemic Planning Domain Definition Language* or *E-PDDL* for short. Consequently, E-PDDL will represent the input for our meta-cognitive architecture.

E-PDDL, as the name clearly shows, inherits its foundations from one of the most adopted action languages: PDDL. The field of planning has seen many representations. For example, in classical planning, there was STRIPS [Fikes and Nilsson, 1971], Action Description Language (ADL) [Pednault, 1994] and SAS+ [Bäckström, 1995] before Planning Domain Description Language (PDDL) [McDermott et al., 1998, Fox and Long, 2003] standardized the notations. Nowadays, planners routinely

use PDDL for problem specification even if they may convert to other representations later for solving efficiency [Helmert, 2009]. PDDL envisages two files, a domain description file which specifies information independent of a problem like predicates and actions, and a problem description file which specifies the initial and goal states. In PDDL, a planning environment is described in terms of objects in the world, predicates that describe relations that hold between these objects, and actions that bring change to the world by manipulating relations. A problem is characterized by an initial state, together with a goal state that the agent wants to transition to, both states specified as configurations of objects. When planning is used for epistemic reasoning, the objects in the problem can be physical (real-world objects) as well as abstract (knowledge and beliefs).

Let us now present E-PDDL, making use of the Coin in the Box domain (Planning Domain 2.1) to better explain some of the features. Let us remark that the syntax has been chosen with the objective of minimizing the difference with standard PDDL while providing a general epistemic input language. We will start by showing the syntax of the problem-*domain*—that contains the general settings of the problem—and then we will illustrate how a problem-*instance*—that contains specific objects, initial conditions, and goals—is characterized. In particular, in Listings 6.1 we present the characterization of the Planning Domain 2.1 problem-*domain* and in Listings 6.2 we present a simple problem-*instance* where it is known that agent `a` has the key and the goal is for `a` to know the coin position. Before proceeding with the description of the E-PDDL syntax we need to introduce the meaning of the operator "`[i]`" where $\mathsf{i} \in \mathcal{AG}$. This operator captures the modal operator $\mathbf{B}_\mathsf{i}$. For example, in Line 10 of Listing 6.1, the formula `[?i](has_key ?i)` reads "agent `i` knows `has_key_i`" where $\mathsf{i} \in \mathcal{AG}$ and the fluent `has_key_i` encodes the fact that `i` has the key. When the operator is of the form "$[\alpha]$" where $\alpha \subseteq \mathcal{AG}$ and $|\alpha| \geq 2$ then it captures the idea of common belief.

Problem *Domain*

First of all, let us note that in Listings 6.1 the actions `signal` and `distract` are omitted to avoid clutter. In fact, these actions are world-altering and, therefore, share a similar structure with the action `open`.

Following the PDDL syntax Fox and Long [2003], we start the problem-*domain* definition by introducing the name and the requirements of the problem (Lines 1 and 2 of Listings 6.1, respectively). We included a new requirement called `:mep` to identify the need for E-PDDL. Lines 4-5 introduce the predicates following the PDDL standard. A small variation is the *object-type* `agent` that does not need to be defined and it is used to define variables that capture the acting agents.

From Line 7 to Line 14 the action `open` is introduced. The action's definition starts with its *name* (Line 7) and its *type* (Line 8). The concept of action type is inherited from $m\mathcal{A}^{\rho}$ and, for now, is restricted to be one among ontic, sensing, or announcement since these are the accepted variations of actions in the MEP community. Alternatively, if the user is using EFP 2.1 and defined some custom event model, their ids could be used. Next, in Line 9 and Line 10, respectively, the action's *parameters* and *preconditions* are defined. The parameters have the same role that they have in PDDL, that is they are used to associate the variables of the action's definition with an *object type*. Similarly, also the field preconditions—identified by any belief formula—follow the standard PDDL meaning. After the preconditions, the action specifies the *effects*. Finally, the last field of the action `open` is about the *observer*s. This field is used to indicate which agents are fully observant, *i.e.*, knows about the execution and the effects of the action. Knowing which agent is observant is useful to derive how the beliefs of the agents are updated after the action is been executed. To better characterize the set of observant agents we introduced the operator `diff` that allows to "isolate" the executor of the action (since the executor needs to be observant and should not depend on other factors). For example, the condition in Lines 12-13 reads as: "the agent `i`, *i.e.*, the executor, is fully observant" and "for every agent `j` $\neq$ `i` if `j is looking` then `j` is fully observant". The same schema is used to define partial observers, the ones that are aware of

the action execution but do not know the results of such action, with the field *p_observers*; an example of partial observability is at Lines 22-23.

```
1  (define (domain coininthebox)
2   (:requirements :strips :negative-preconditions :mep)
3
4   (:predicates (opened) (tail)
5                (has_key ?i - agent) (looking ?i - agent)))
6
7   (:action open
8     :act_type      ontic
9     :parameters    (?i - agent)
10    :precondition  ([?i](has_key ?i))
11    :effect        (opened)
12    :observers     (and (?i) (forall (diff (?j - agent)(?i))
13                   (when (looking ?j) (?j))))
14  )
15
16  (:action peek
17    :act_type      sensing
18    :parameters    (?i - agent)
19    :precondition  (and ([?i](opened)) ([?i](looking ?i)))
20    :effect        (tail)
21    :observers     (?i)
22    :p_observers   (forall (diff (?j - agent)(?i))
23                   (when (looking ?j) (?j)))
24  )
25
26  (:action announce
27    :act_type      announcement
28    :parameters    (?i - agent)
29    :precondition  ([?i](tail))
30    :effect        (tail)
31    :observers     (and (?i) (forall (diff (?j - agent)(?i))
32                   (when (looking ?j) (?j))))
33  )
34 )
```

Listing 6.1: E-PDDL Coin in the Box problem-*domain*.

The introduced fields are tailored for *implicit belief update*, namely a transition function that automatically updates the e-states without having to know the list of belief formulae that have been verified or negated. This is the case of $m\mathcal{A}^{\rho}$, which derives how to structurally update the e-state knowing action type, observability relations, and which properties of the world have been modified. Later in this section, we will also explain how, with the presented syntax, it is possible to generate a valid input also for those planners that need the effects of the action to be completely explicit.

Problem *Instance*

In Listings 6.2 we present an example of an E-PDDL problem-*instance*. In Line 1 and in Line 2 the problem-*instance* name (*i.e.*, `toyinstance`) and the related problem-*domain* name are defined, respectively.

Next, in Line 3, the *object type* `agent` values are defined. In this particular instance, we defined three agents `a`, `b` and `c` as in Planning Domain 2.1.

Following, in Line 4, the `depth` is specified. The concept of depth of a belief formula is used to identify the number of *nested* epistemic operators. For example, given two agents `i` and `j` the belief formula $\mathbf{B}_{\mathsf{i}}(\varphi)$ has depth 1 while $\mathbf{B}_{\mathsf{i}}(\mathbf{C}_{\mathsf{i,j}}(\varphi))$ has depth 2. This field is introduced to accommodate the need for certain planners, *e.g.*, RP-MEP, to limit the depth of the belief formulae. RP-MEP relies on grounding the formulae into classical planning "facts" that without bound on these formulae could be infinite. On the other hand, the limit on depth is ignored by the planners, *e.g.*, EFP, that reason directly on epistemic states.

Lines 5-12 present the belief formulae that describe the initial state. The formulae are considered to be in conjunction with each other. The *initial conditions* only require to specify when a fluent is true and consider false whichever fluent is not specified (Line 5). Moreover, the initial conditions are also used to specify what is known in the initial state (Line 6-12). While the belief formulae in this field can be of any type, let us remark that Son et al. [2014] demonstrated that to create a finite number of epistemic states from a set of formulae, this set must respect a finitary **S5** logic and therefore the beliefs must be expressed in terms of common belief. This means that if the initial conditions do not comply with a finitary **S5** logic the planners that construct the initial epistemic state from the given specification may not work.

Finally, in Line 13 the conjunction of belief formulae that represent the goals is defined.

```
1  (define (problem toyinstance)
2   (:domain coininthebox)
3   (:agent a b c)
4   (:depth 2)
5   (:init (tails) (has_key a) (looking a)
6          ([a b c](has_key a))
7          ([a b c](not (has_key b)))
8          ([a b c](not (has_key c))))
9          ([a b c](not (opened)))
10         ([a b c](looking a))
11         ([a b c](not (looking b)))
12         ([a b c](not (looking c))))
13  (:goal ([a](tails)))
14 )
```

Listing 6.2: E-PDDL Coin in the Box problem-*instance*.

From Implicit to Explicit Belief Update

Since we tailored E-PDDL syntax to represent actions with implicit belief update we need to explain how E-PDDL itself is a suitable language also for those planners, *e.g.*, RP-MEP, that need the belief update to be explicit. That is, we need a standard way of deriving the explicit agents' belief update from an E-PDDL action description. While deriving explicit belief update is not a method that is embedded in the language itself in what follows we propose the strategy that we adopted in our implemented parser.

The strategy that we adopted in our parser is based on the transition function by Baral et al. [2015] where the idea of agents' observability is used to derive consistent agents' beliefs about the actions' effects and/or execution. In what follows we present a *belief derivation schema* that, starting from the agents' observability, generates the explicit belief update related to a single action. These updates generate all the belief-chains of finite length ℓ where $\ell \in \{0, \ldots, \mathtt{d}\}$ and `d` is the value assigned to the field `:depth` in the problem-*instance* (in Listings 6.2 $\mathtt{d} = 2$). Namely, we will have all the following chains:

- a fully observant knows the action's effect ($\ell = 1$);
- a fully observant knows that another fully observant knows the action's effect ($\ell = 2$);

- a fully observant knows the chains of length 2 ($\ell = 3$);
- and so on until we have that $\ell = \mathtt{d}$.

Moreover, when partially observant agents are defined we also need to take into consideration their perspective on the belief update. To do that we will need to automatically generate the following belief-chains (still limited by the given depth):

- a partially observant knows that any chain of fully observant agents knows the action's effect ($\ell = 2$); and
- any chain, with $\ell \leq \mathtt{d} - 2$, of fully/partially observant knows the chain of beliefs presented in the previous point.

To better integrate the explicit belief update, we decided to incorporate E-PDDL with an extra, non-mandatory field for the actions' specification. This field, identified by `:exp_effect` can be used to identify the explicit belief update of the action by using an arbitrary belief formula. Let us note that when this field is defined it will completely override the automatically derived belief update for the planners that make use of explicit belief update, *e.g.*, RP-MEP[2]. On the other hand, planners that make use of a full-fledged epistemic transition function, *e.g.*, EFP, will ignore the `:exp_effect` field.

6.3.2 The Overall Architecture

Thanks to E-PDDL we are now able to define MEP problems in a standardized way. Next, we need to take the given input, in E-PDDL, and return a solution to it using the **MC** module formalized above. This implementation[3] introduces an early implementation of a `System-1` strategy to solve epistemic problems and exploits already existing planners as `System-2` solvers. In particular, the former makes heavy use of already found plans, returning as a solution the plan that verifies the

[2]We envisioned this functionality as a way of explicitly providing all the needed effects of the actions. Nonetheless, in the particular case of RP-MEP, the "experienced" user could just define the base effects of the actions that would be later compiled by RP-MEP into ancillary effects.

[3]Available at `https://github.com/FrancescoFabiano/MetacognitiveEpistemicPlanning`.

major number of sub-goals in the domain. This is still a rudimentary approach and will certainly be improved over future iterations, but still allows to experiment with the **MC** structure. The `System-2` planners, used as black boxes, are instead the planners RP-MEP [Muise et al., 2015][4], and EFP 2.1.

Let us now present a high-level description of the architecture.

- Initially, the tool receives three input files: a domain description in E-PDDL, a problem instance in E-PDDL, and a *context* file. The last file contains meta-data, *e.g.*, resources availability, accuracy required, that emulate the limits represented by the environment.

- Then the architecture, emulating the **MC-1** module, checks whether there is enough experience to retrieve a plan, from past instances, that solves the problem respecting the given constraints. If such a plan exists, it is returned as a solution.

- Otherwise, a simplified version of **MC-2** is engaged. This part of the architecture analyzes the problem and, after checking the maximum depth and the presence of dynamic common belief, selects the best option between **Pla-1** (RP-MEP) and **Pla-2** (EFP 2.1).

- After selecting the best approach for the given problem, the tool evaluates the problem difficulty and derives the expected resource consumption (with respect to the selected planner).

- Then, the architecture checks if the solving process is within the constraints. If it is not, the solution from the `System-1` solver (if exists) is adopted. On the other hand, if the estimated time is within the given constraints, the problem is: *(i)* translated in the language "understood" by the selected `System-2` solver thanks to our E-PDDL parser; *(ii)* solved by either **Pla-1** or **Pla-2**; and *(iii)* then validated and saved alongside its solution to increase the system's experience.

[4]Available at `https://github.com/QuMuLab/pdkb-planning`.

[...] ἔοικα γοῦν τούτου γε σμικρῷ τινι αὐτῷ τούτῳ σοφώτερος εἶναι, ὅτι ἃ μὴ οἶδα οὐδὲ οἴομαι εἰδέναι.

I neither know nor think I know. (Paraphrase)

— Socrates
in Plato, *Apology* [21d]

7 Conclusion

In this dissertation, we presented our research efforts in formalizing and developing a general and flexible Multi-agent Epistemic Planning environment. The final goal of this thesis is to deliver an instrument that can be exploited as a basis for future research on the MEP setting.

We started by illustrating a new epistemic state representation that, alongside a new and optimized transition function, allowed us to design a comprehensive epistemic solver with state-of-the-art performances. Doing so, we presented $m\mathcal{A}^{\rho}$, an action language for MEP based on possibilities—a non-well-founded data structure. $m\mathcal{A}^{\rho}$ imitates its predecessor $m\mathcal{A}^{*}$ in defining three types of actions: world-altering, sensing, and announcements. While these action types allow to describe a vast range of domains, we decided to enrich $m\mathcal{A}^{\rho}$—and consequently, our solver based on it—in order to capture an even wider spectrum of real-world scenarios. We, therefore, decided to start by defining one of the fundamental concepts that are linked to information flows: the idea of trust. This permitted to define how agents treat incoming information considering the source. We then envisioned a way to expand our epistemic language even further. We associated each agent to a specific *attitude* that varies depending on the world configuration and the information source. Attitudes enrich the domain description by defining how the agents handle the various information exchanges. After implementing and validating all the previous

expansions, we formalized a general framework that allows the user to define custom action types. Thanks to this final step, it is possible to tailor action types without limitations, making our planner a tool capable of handling all the various epistemic nuances. Finally, we implemented an initial version of a two phases architecture that, inspired by a famous cognitive theory, exploits diverse techniques to optimize the resolution of MEP problems. Once again, we hope that this architecture can be useful to other researchers who may also complement it with their tools.

While the functionalities described above have been implemented, we believe that there are still a lot of different research directions that need to be analyzed in the MEP setting. For example, we believe that it is paramount to study distributed versions of epistemic solvers. This would allow for a better characterization of multi-agent scenarios allowing, for example, to better capture secrecy and to make the planning process more realistic. Another important factor that we did not explore during our research is the concept of non-deterministic actions. While these could be "easily" addressed at the *search-space level*, we believe that it would be much more appropriate to address them within the single e-state update. Several other improvements in the formalization could be devised, and we hope that our framework would help in doing so in future studies. Finally, considering the planner, we feel like there is still much work to be done. In fact, an important issue for MEP solvers is their poor scalability. In this dissertation, we put most of our efforts into investigating the foundation of the problem rather than optimizing what already existed. Nonetheless, having tools that, most of the time, have not acceptable performances limit the proliferation of the solvers themselves. That is why we believe a very important future work is to focus on the optimization of EFP, making it suitable for real-world tasks. Such optimizations could derive from several directions: implementation of heuristics (formalized in this thesis), use of parallelism, adoption of symbolic e-state representations, and so on. Furthermore, we believe that devising a simple user interface for the aforementioned tools, especially EFP, will make them more approachable.

Appendices

Cred'io ch'ei credette ch'io credesse [...]

I believe he believed that I believed that [...]

— Dante Alighieri
Inferno, XIII, 25-26

A
Propositions Proofs

Contents

A.1 Preliminary Definitions

Before starting with the proofs we need to introduce some terminology that will help us to avoid unnecessary clutter. In particular, let a Domain D, a $\mathsf{p} \in \mathcal{S}$ where $\mathcal{S}$ is the set of all the possibilities reachable from $D(\varphi_{ini})$ with a finite sequence of action instances, and the set of agents $\mathcal{AG} \subseteq D(\mathcal{AG})$ be given. The operator $\mathcal{B}^{\mathsf{p}}_{\mathcal{AG}}$ captures *all the reachable possibilities for* $\mathcal{AG}$ *given a starting possibility* p. Let us describe now how this operator can be used to represent the notions of *(i)* agents' belief; *(ii)* common belief; and *(iii)* nested beliefs.

Agents Beliefs Representation To link the operator introduced above with the concept of belief let us start with the case where the set of agents $\mathcal{AG}$ contains only one element i, *i.e.*, $\mathcal{AG} = \{\mathsf{i}\}$. We, therefore, use $\mathcal{B}_{\mathsf{i}}^{\mathsf{p}}$ to identify the set of all the possibilities that i, starting from the possibility p, cannot distinguish. The construction of the set identified by $\mathcal{B}_{\mathsf{i}}^{\mathsf{p}}$ is procedural and it is done by applying the operator $(\mathcal{B}_{\mathsf{i}}^{\mathsf{p}})^k$, with $k \in \mathbb{N}$, until a *fixed point* is found. The operator $(\mathcal{B}_{\mathsf{i}}^{\mathsf{p}})^k$ is defined as follows:

$$(\mathcal{B}_{\mathsf{i}}^{\mathsf{p}})^k = \begin{cases} \mathsf{p}(\mathsf{i}) & \text{if } k = 0 \\ \{\mathsf{q} \mid (\exists \mathsf{u} \in (\mathcal{B}_{\mathsf{i}}^{\mathsf{p}})^{k-1})(\mathsf{q} \in \mathsf{u}(\mathsf{i}))\} & \text{if } k \geq 1 \end{cases}$$

Finally, we can define $\mathcal{B}_{\mathsf{i}}^{\mathsf{p}} = \bigcup_{k \geq 1} (\mathcal{B}_{\mathsf{i}}^{\mathsf{p}})^k$. It is easy to see that this is equivalent to the set of possibilities reached by the operator $\mathbf{B}_{\mathsf{i}}$ starting from p and, therefore, that it represents the beliefs of i in p.

Let us note that the fixed point of the succession $(\mathcal{B}_{\mathcal{AG}}^{\mathcal{S}})^k$ is reached in a finite number of iterations. This is because:

- $(\mathcal{B}_{\mathcal{AG}}^{\mathcal{S}})^k$ is monotonic; namely $(\mathcal{B}_{\mathcal{AG}}^{\mathcal{S}})^k \subseteq (\mathcal{B}_{\mathcal{AG}}^{\mathcal{S}})^{k+1}$ with $k \in \mathbb{N}$ (Lemma A.1); and

- the set $\mathcal{S}$ of all the possibilities reached by applying a finite sequence of action instances Δ to a given possibility p has a finite number of elements (Lemma A.2).

Common Belief Representation Now, similarly to the single-agent case, we can define the set $\mathcal{B}_{\mathcal{AG}}^{\mathsf{p}}$. This represents the *common belief* of $\mathcal{AG}$ ($\mathbf{C}_{\mathcal{AG}}$) starting from p. As before we introduce the operator $(\mathcal{B}_{\mathcal{AG}}^{\mathsf{p}})^k$ of which the fixed point will result in $\mathcal{B}_{\mathcal{AG}}^{\mathsf{p}}$.

$$(\mathcal{B}_{\mathcal{AG}}^{\mathsf{p}})^k = \begin{cases} \bigcup_{\mathsf{i} \in \mathcal{AG}} \mathsf{p}(\mathsf{i}) & \text{if } k = 0 \\ \{\mathsf{q} \mid (\exists \mathsf{u} \in (\mathcal{B}_{\mathcal{AG}}^{\mathsf{p}})^{k-1})(\mathsf{q} \in \bigcup_{\mathsf{i} \in \mathcal{AG}} \mathsf{u}(\mathsf{i}))\} & \text{if } k \geq 1 \end{cases}$$

Nested Belief Representation We can also express the concept of *nested belief* in a more compact way. Let two sets of agents $\mathcal{AG}_1 \subseteq D(\mathcal{AG}), \mathcal{AG}_2 \subseteq D(\mathcal{AG})$ be given; the set of possibilities reachable by applying $\mathbf{C}_{\mathcal{AG}_1}\mathbf{C}_{\mathcal{AG}_2}$ starting from p is:

$$\mathcal{B}^{\mathsf{p}}_{\mathcal{AG}_1,\mathcal{AG}_2} = \{\mathsf{q} \mid (\exists \mathsf{r} \in \mathcal{B}^{\mathsf{p}}_{\mathcal{AG}_1})(\mathsf{q} \in \mathcal{B}^{\mathsf{r}}_{\mathcal{AG}_2})\}$$

Let us note that, when $\mathcal{AG}_1$ or $\mathcal{AG}_2$ contains only one agent i, $\mathbf{C}_\mathsf{i}$, and $\mathbf{B}_\mathsf{i}$ are equal.

Lemma 1.1: Operator $\mathcal{B}^{\mathcal{S}}_{\mathcal{AG}}$ monotony

The sequence $(\mathcal{B}^{\mathcal{S}}_{\mathcal{AG}})$ is monotonic; meaning that, for every $k \in \mathbb{N}$, $(\mathcal{B}^{\mathcal{S}}_{\mathcal{AG}})^k \subseteq (\mathcal{B}^{\mathcal{S}}_{\mathcal{AG}})^{k+1}$.

Proof of Lemma A.1 Without losing generality let a possibility p and an agent i be given. To prove the monotonicity of $(\mathcal{B}^{\mathsf{p}}_\mathsf{i})$ we start by recalling that:

$(\mathcal{B}^{\mathsf{p}}_\mathsf{i})^k = \{\mathsf{q} \mid (\exists \mathsf{u} \in (\mathcal{B}^{\mathsf{p}}_\mathsf{i})^{k-1})(\mathsf{q} \in \mathsf{u}(\mathsf{i}))\}$.

By construction, each possibility respects the **KD45** logic (Table 1.1) and, therefore, some structural constraints. In particular, to comply with axioms **4** and **5**, if a possibility $\mathsf{q} \in \mathsf{p}(\mathsf{i})$ then $\mathsf{q} \in \mathsf{q}(\mathsf{i})$. In terms of our sequence, this translates into *if a possibility* $\mathsf{q} \in (\mathcal{B}^{\mathsf{p}}_\mathsf{i})^{k-1}$ *then* $\mathsf{q} \in (\mathcal{B}^{\mathsf{p}}_\mathsf{i})^k$.

It is easy to see that this property ensures that the agent's reachability function respect introspection. That is; when an agent reaches q she/he has to "know" that her/him-self considers q possible. Thanks to this property we can now infer that each iteration of the sequence $(\mathcal{B}^{\mathsf{p}}_\mathsf{i})^k$ contains at least $(\mathcal{B}^{\mathsf{p}}_\mathsf{i})^{k-1}$ and, therefore, that the sequence $(\mathcal{B}^{\mathcal{S}}_{\mathcal{AG}})$ is monotonic. □

Lemma 1.2: States Size Finiteness

Given a finite action instances sequence Δ—namely a plan*—and a starting point p with a finite number of possible worlds,* i.e., *$|\bigcup_{\mathsf{i} \in D(\mathcal{AG})} \mathsf{p}(\mathsf{i})| = n$, the set $\mathcal{S}$ of all the possibilities generated by applying Δ to p has a finite number of elements.*

Proof of Lemma A.2 Following the definition of the transition function of $m\mathcal{A}^{\rho}$ (Definition 2.12) we can determine an upper bound for the number of new possibilities generated after the application of an action instance and, therefore, of an action instances sequence. In particular, from a given possibility p such that $|\mathcal{B}^{\mathsf{p}}_{\mathcal{AG}}| = n$ (where $\mathcal{AG}$ is the set of all the agents) the cardinality of the set $\mathcal{B}^{\mathsf{p}'}_{\mathcal{AG}}$ will be, at most, equal to $2n$. That is because:

- when an *ontic* action is executed each possibility $\in |\mathcal{B}^{\mathsf{p}}_{\mathcal{AG}}|$ can be either updated—if reached by a fully observant agent—or kept unchanged—if reached by an oblivious agent. This means that an upper bound to the size of $\mathcal{B}^{\mathsf{p}'}_{\mathcal{AG}}$ in case of an ontic action execution is $2n$ where only the updated possibilities (n) are new elements of $\mathcal{S}$.
- The case with *sensing* and *announcement* actions is similar.

This identifies $2n$ as the upper bound for the growth of a state size and for the generation of new possibilities after an action execution. Therefore, given the size n of the initial state and the length of the action sequence l we can conclude that $|\mathcal{S}| \leq (n \times 2^l)$ that is indeed finite. □

A.2 Proofs of Propositions 2.3 to 2.5

Let us prove the properties illustrated in Propositions 2.3 to 2.5.

As before, in the following proofs, we will use p' instead of $\Phi(\mathsf{a}, \mathsf{p})$ to avoid unnecessary clutter when possible.

Proof of Proposition 2.3 Let us prove each item of Proposition 2.3 separately:

(1) Assuming that action a is executable in u we have that $\mathsf{u} \models \psi$. This means that:

- If $\mathsf{u} \models \mathbf{B}_{\mathsf{x}}(\psi)$ we have that $\forall \mathsf{p} \in \mathcal{B}^{\mathsf{u}}_{\mathsf{x}}\ \mathsf{p} \models \psi$; this is because $\mathcal{B}^{\mathsf{u}}_{\mathsf{x}}$ represents the set of possibilities reachable by $\mathbf{B}_{\mathsf{x}}$ starting from u.
- In particular we are interested in the set of possibilities reachable by $\mathbf{B}_{\mathsf{x}}$ starting from u', *i.e.*, $\mathcal{B}^{\mathsf{u}'}_{\mathsf{x}} = \{\mathsf{p}' \mid (\exists \mathsf{p} \in \mathcal{B}^{\mathsf{u}}_{\mathsf{x}})(\mathsf{p}' = \Phi(\mathsf{p}, \mathsf{a}))\}$.
- Following Definition 2.12, we also know that—being $\mathsf{x} \in \mathbf{F}_{\mathsf{a}}$—if $\ell = \mathtt{f}^a$ then $e(\mathsf{a}, \mathsf{u}) = \{\mathtt{f}\}$ and therefore $\mathsf{p}'(\mathtt{f}) = 1\ \forall \mathsf{p}' \in \mathcal{B}^{\mathsf{u}'}_{\mathsf{x}}$.
- From this last step we can conclude that every element of $\mathcal{B}^{\mathsf{u}'}_{\mathsf{i}}$ entails $\mathtt{f}$.

- As said previously $\mathcal{B}^{\mathsf{u}'}_{\mathsf{x}}$ represents $\mathbf{B}_{\mathsf{x}}$ starting from u'.
- It is easy to see that if every element in $\mathcal{B}^{\mathsf{u}'}_{\mathsf{x}}$ entails $\mathtt{f}$, then $\mathsf{u}' \models \mathbf{B}_{\mathsf{x}}(f)$.

(2) As in the previous item, we assume action a to be executable in u meaning that:

- If $\mathsf{u} \models \mathbf{B}_{\mathsf{y}}(\varphi)$ we have that every $\mathsf{p} \in \mathcal{B}^{\mathsf{u}}_{\mathsf{y}}$ entails φ.
- From Definition 2.12 when $\mathsf{y} \in \mathbf{O}_{\mathsf{a}}$ for each possibility $\mathsf{p} \in \mathcal{B}^{\mathsf{u}}_{\mathsf{y}}$ $\mathsf{p}(\mathsf{y}) = \mathsf{p}'(\mathsf{y})$ it is easy to see that $\mathcal{B}^{\mathsf{u}}_{\mathsf{y}} \equiv \mathcal{B}^{\mathsf{u}'}_{\mathsf{y}}$.
- Given that the two sets of possibilities are the same, it means that the reachability functions that they represent are the same.
- Being the two functions the same it means that $\forall \varphi \in D$ $\mathsf{u} \models \mathbf{B}_{\mathsf{y}}(\varphi)$ *iff* $\mathsf{u}' \models \mathbf{B}_{\mathsf{y}}(\varphi)$.

(3) Again we assume the executability of the action a and we consider $\mathsf{x} \in \mathbf{F}_{\mathsf{a}}$ and $\mathsf{y} \in \mathbf{O}_{\mathsf{a}}$:

- Being $\mathsf{y} \in \mathbf{O}_{\mathsf{a}}$, from Definition 2.12, we know that $\mathsf{p}(\mathsf{y}) = \mathsf{p}'(\mathsf{y})$ such that $\mathsf{p} \in \mathcal{B}^{\mathsf{u}}_{\mathsf{x}}$ and p' is its updated version $\in \mathcal{B}^{\mathsf{u}'}_{\mathsf{x}}$.
- This means that for every element in $\mathcal{B}^{\mathsf{u}}_{\mathsf{x}}$ we have an updated version that has the same reachability function for the agent y.
- Then it is easy to see that $\mathcal{B}^{\mathsf{u}}_{\mathsf{x},\mathsf{y}} \equiv \mathcal{B}^{\mathsf{u}'}_{\mathsf{x},\mathsf{y}}$ and therefore that these two sets contain the same possibilities.
- As already said in Item **(2)** when two sets of possibilities are the same they entail the same formulae.
- Therefore we can conclude that if $\mathsf{u} \models \mathbf{B}_{\mathsf{x}}(\mathbf{B}_{\mathsf{y}}(\varphi))$ then $\mathsf{u}' \models \mathbf{B}_{\mathsf{x}}(\mathbf{B}_{\mathsf{y}}(\varphi))$

□

[a] The case where a **causes** $\neg\mathtt{f}$ is similar and, therefore, is omitted here.

Proof of Proposition 2.4 Once again, let us prove each item separately:

(1) In the following we prove Item **(1)**. Being the proof for Item **(2)** similar we will omit it for the sake of readability.

- First of all we identify the set of all the possibilities reached by the *fully observant* agents in u as $\mathcal{B}^{\mathsf{u}}_{\mathbf{F}_\mathsf{a}}$ and we remind that, as shown in Paragraph **Common Belief Representation**, this set corresponds to the possibilities reached by $\mathbf{C}_{\mathbf{F}_\mathsf{a}}$;
- We recall that, by hypothesis, $\mathsf{u} \models \mathtt{f}$ and therefore $e(\mathsf{a}, \mathsf{u}) = \{\mathtt{f}\}$.
- We then calculate $\mathcal{B}^{\mathsf{u}'}_{\mathbf{F}_\mathsf{a}}$ that, following Definition 2.12, contains only possibilities p' such that $\mathsf{p}'(\mathtt{f}) = 1$.
- This means that $\forall \mathsf{p}' \in \mathcal{B}^{\mathsf{u}'}_{\mathbf{F}_\mathsf{a}}$ we have that $\mathsf{p}' \models \mathtt{f}$.
- As shown in Item **(1)** of Proposition 2.3, given that this set contains only the possibilities that entail $\mathtt{f}$ we can derive that $\mathcal{B}^{\mathsf{u}'}_{\mathbf{F}_\mathsf{a}} \models \mathtt{f}$.
- Finally, as the set $\mathbf{C}_{\mathbf{F}_\mathsf{a}} \equiv \mathcal{B}^{\mathsf{u}'}_{\mathbf{F}_\mathsf{a}}$, we have that $\mathbf{C}_{\mathbf{F}_\mathsf{a}} \models \mathtt{f}$.

(2) The proof of this item is similar to the one presented in Item **(1)** and it is omitted for the sake of readability.

(3) Once again we identify the set of the possibilities reachable by *partial observant* agents with $\mathcal{B}^{\mathsf{u}}_{\mathbf{P}_\mathsf{a}}$. We also remind that this set is equal to $\mathbf{C}_{\mathbf{P}_\mathsf{a}}$ in u.

- Now to calculate $\mathcal{B}^{\mathsf{u}'}_{\mathbf{P}_\mathsf{a}}$, following Definition 2.12, we apply "$\Phi(\mathsf{a}, \mathsf{u})$" to every element of $\mathcal{B}^{\mathsf{u}}_{\mathbf{P}_\mathsf{a}}$.
- To simplify the proof let us redefine the partially observant agents' belief update for epistemic actions in the following way:

$$\mathsf{u}'(\mathsf{i}) = \begin{cases} \bigcup\limits_{\mathsf{w} \in \mathsf{u}(\mathsf{i})} \Phi(\mathsf{a}, \mathsf{w}) & \text{if } \mathsf{i} \in \mathcal{AG}, \mathsf{i} \in \mathbf{P}_\mathsf{a} \text{ and } e(\mathsf{a}, \mathsf{u}) = e(\mathsf{a}, \mathsf{w}) \\ \bigcup\limits_{\mathsf{w} \in \mathsf{u}(\mathsf{i})} \Phi(\mathsf{a}, \mathsf{w}) & \text{if } \mathsf{i} \in \mathcal{AG}, \mathsf{i} \in \mathbf{P}_\mathsf{a} \text{ and } e(\mathsf{a}, \mathsf{u}) \neq e(\mathsf{a}, \mathsf{w}) \end{cases}$$

Where $\mathsf{i} \in \mathbf{P}_\mathsf{a}$

- It is easy to identify two disjoint subsets $\mathcal{B}^1_{\mathbf{P}_a}$ and $\mathcal{B}^2_{\mathbf{P}_a}$ of $\mathcal{B}^{\mathsf{u}'}_{\mathbf{P}_a}$ that contains only possibility such that:
  - $\mathcal{B}^1_{\mathbf{P}_a} \models e(\mathsf{a}, \mathsf{u})$;
  - $\mathcal{B}^2_{\mathbf{P}_a} \not\models e(\mathsf{a}, \mathsf{u})$;
  - $(\mathcal{B}^1_{\mathbf{P}_a} \cup \mathcal{B}^2_{\mathbf{P}_a}) \equiv \mathcal{B}^{\mathsf{u}'}_{\mathbf{P}_a}$; and
  - $(\mathcal{B}^1_{\mathbf{P}_a} \cap \mathcal{B}^2_{\mathbf{P}_a}) \equiv \emptyset$.

- From these two sets we can now construct the sets $\mathcal{B}^1_{\mathbf{P}_a,\mathbf{F}_a}$ and $\mathcal{B}^2_{\mathbf{P}_a,\mathbf{F}_a}$ that are simply the set of possibilities reachable from the *fully observant* agents starting from $\mathcal{B}^1_{\mathbf{P}_a}$ and $\mathcal{B}^2_{\mathbf{P}_a}$, respectively.

- Given that the set $\mathcal{B}^1_{\mathbf{P}_a,\mathbf{F}_a}$ resulted from the application of the transition function from the point of view of fully observant agents, we know from Item **(1)** of Proposition 2.3 that for $\forall \mathsf{p} \in \mathcal{B}^1_{\mathbf{P}_a,\mathbf{F}_a}, \mathsf{p} \models \mathtt{f}$.

- This implies that $\mathcal{B}^1_{\mathbf{P}_a,\mathbf{F}_a}$ reaches only possibilities where the interpretation of $\mathtt{f}$ is true and similarly in $\mathcal{B}^2_{\mathbf{P}_a,\mathbf{F}_a}$ only possibilities where the interpretation of $\mathtt{f}$ is false.

- This means that $\mathcal{B}^1_{\mathbf{P}_a,\mathbf{F}_a} \models \mathtt{f}$ and $\mathcal{B}^2_{\mathbf{P}_a,\mathbf{F}_a} \models \neg\mathtt{f}$.

- It is easy to see then that $\mathcal{B}^1_{\mathbf{P}_a} \models \mathbf{C}_{\mathbf{F}_a}\mathtt{f}$ being $\mathcal{B}^1_{\mathbf{P}_a,\mathbf{F}_a} = \{\mathsf{p} \mid \mathsf{p} \in \bigcup_{\mathsf{q}\in\mathcal{B}^1_{\mathbf{P}_a}} \mathsf{q}(\mathbf{F}_a)\}$ (and similarly $\mathcal{B}^2_{\mathbf{P}_a} \models \mathbf{C}_{\mathbf{F}_a}\neg\mathtt{f}$).

- Finally being $\mathcal{B}^{\mathsf{u}'}_{\mathbf{P}_a} = \mathcal{B}^1_{\mathbf{P}_a} \cup \mathcal{B}^2_{\mathbf{P}_a}$ we can conclude that $\mathcal{B}^{\mathsf{u}'}_{\mathbf{P}_a} \models \mathbf{C}_{\mathbf{F}_a}\mathtt{f} \vee \mathbf{C}_{\mathbf{F}_a}\neg\mathtt{f}^a$ and therefore $\mathsf{u}' \models \mathbf{C}_{\mathbf{P}_a}(\mathbf{C}_{\mathbf{F}_a}\mathtt{f} \vee \mathbf{C}_{\mathbf{F}_a}\neg\mathtt{f})$.

(4) To prove this item we will make use of the properties proved in previous Items.

- As said in the Paragraph **Nested Belief Representation**, we know that $\mathcal{B}^{\mathsf{u}}_{\mathbf{F}_a,\mathbf{P}_a}$ corresponds with the set of possibilities identified by $\mathbf{C}_{\mathbf{F}_a}\mathbf{C}_{\mathbf{P}_a}$ and it is also equal to $\{\mathsf{p} \mid (\exists \mathsf{q} \in \mathcal{B}^{\mathsf{u}}_{\mathbf{P}_a})(\mathsf{p} \in \bigcup_{\mathsf{i}\in\mathbf{F}_a} \mathsf{q}(\mathsf{i}))\}$.

- Now to calculate $\mathcal{B}^{\mathsf{u}'}_{\mathbf{F}_a}$ we apply Definition 2.12 to every element of $\mathcal{B}^{\mathsf{u}}_{\mathbf{F}_a}$. This means that $\mathcal{B}^{\mathsf{u}'}_{\mathbf{F}_a} = \{\mathsf{p}' \mid (\exists \mathsf{p} \in \mathcal{B}^{\mathsf{u}}_{\mathbf{F}_a})(\mathsf{p}' = \Phi(\mathsf{a}, \mathsf{p}))\}$.

- We then want to calculate the set $\{\mathsf{p}' \mid (\exists \mathsf{q}' \in \mathcal{B}^{\mathsf{u}'}_{\mathbf{F}_a})(\mathsf{p}' \in \bigcup_{\mathsf{i}\in\mathbf{P}_a} \mathsf{q}'(\mathsf{i}))\}$.

- To calculate the "point of view" of the partially observants with respect to the fully observants we apply Definition 2.12 to all the elements of

$\{\mathsf{p} \mid (\exists \mathsf{q}' \in \mathcal{B}^{\mathsf{u}'}_{\mathbf{F}_\mathsf{a}})(\mathsf{p} \in \mathcal{B}^{\mathsf{q}}_{\mathbf{P}_\mathsf{a}})\}$.

- It is easy to see that the resulting set is $\{\mathsf{p}' \mid (\exists \mathsf{q}' \in \mathcal{B}^{\mathsf{u}'}_{\mathbf{F}_\mathsf{a}})(\mathsf{p}' \in \bigcup_{\mathsf{i} \in \mathbf{P}_\mathsf{a}} \mathsf{q}'(\mathsf{i}))\} \equiv \mathcal{B}^{\mathsf{u}'}_{\mathbf{F}_\mathsf{a}, \mathbf{P}_\mathsf{a}}$.
- We showed, in the previous item, that the set $\mathcal{B}^{\mathsf{u}'}_{\mathbf{P}_\mathsf{a}}$ entails $\mathbf{C}_{\mathbf{F}_\mathsf{a}}\mathtt{f} \vee \mathbf{C}_{\mathbf{F}_\mathsf{a}}\neg\mathtt{f}$.
- This means that $\mathcal{B}^{\mathsf{u}'}_{\mathbf{F}_\mathsf{a}, \mathbf{P}_\mathsf{a}} \models (\mathbf{C}_{\mathbf{P}_\mathsf{a}}((\mathbf{C}_{\mathbf{F}_\mathsf{a}}\mathtt{f} \vee \mathbf{C}_{\mathbf{F}_\mathsf{a}}\neg\mathtt{f}))$ and therefore, following what said in Paragraph **Nested Belief Representation**, $\mathsf{u}' \models \mathbf{C}_{\mathbf{F}_\mathsf{a}}(\mathbf{C}_{\mathbf{P}_\mathsf{a}}(\mathbf{C}_{\mathbf{F}_\mathsf{a}}\mathtt{f} \vee \mathbf{C}_{\mathbf{F}_\mathsf{a}}\neg\mathtt{f}))$.

(5)–**(6)** The proofs for the fifth and sixth items are similar to the ones presented in Item **(2)** and Item **(3)** of Proposition 2.3 respectively and is therefore omitted.

□

[a] The two sets are completely disjoint as one only contains possibilities that entail `f` while the other only possibilities that do not. This means that that does not exist any fully-observant-edge between possibilities that belongs in two different sets.

Proof of Proposition 2.5 The proof of this proposition is very similar to the proof of Proposition 2.4 and it is, therefore, omitted for the sake of the presentation. □

A.3 Proofs of Propositions 3.1 and 3.2

Let us provide the formal proof that the properties presented in Propositions 3.1 and 3.2. Most of the properties are shared between *un*-trustworthy announcement and *mis*-trustworthy announcement; and their proof of correctness is the same independently of the announcement type we are considering. For the sake of readability, we will only report the proofs of Items **(1)** to **(7)** considering the *un*-trustworthy announcement, while we explore the remaining properties considering the specific action type.

In the following proofs, without loss of generality, we will consider that $\mathsf{u} \models \phi$. The case when $\mathsf{u} \models \neg\phi$ is a straightforward adaptation and it is, therefore, omitted. Moreover, let us consider the case when the executability conditions of the action a are met. In the case when these conditions are not satisfied, the action update is not executed and therefore does not need to be proved.

A.3.1 Updated States Size Finiteness

Before providing the formal proof let us slightly modify the proof of Lemma A.2 so that it considers also the new actions. In particular, in what follows we prove the e-state finiteness considering also the *un*-trustworthy announcement and the *mis*-trustworthy announcement.

Proof of Lemma A.2 (updated) Following the definition of the transition function of $m\mathcal{A}^{\rho}$ (Definition 2.12) enriched with the actions *un*-trustworthy announcement and *mis*-trustworthy announcement, we can determine an upper bound to the number of new possibilities generated after the application of an action instance and, furthermore, of an action instances sequence. In particular, from a given possibility p such that $|\mathcal{B}^{\mathsf{p}}_{\mathcal{AG}}| = n$ (where $\mathcal{AG}$ is the set of all the agents) the cardinality of the set $\mathcal{B}^{\mathsf{p}'}_{\mathcal{AG}}$ will be, at most, equal to $3n$. That is because:

- when an *ontic* action is executed each possibility $\in |\mathcal{B}^{\mathsf{p}}_{\mathcal{AG}}|$ can be either updated—if reached by a fully observant agent—or kept unchanged—if reached by an oblivious agent. This means that an upper bound to the size of $\mathcal{B}^{\mathsf{p}'}_{\mathcal{AG}}$ in case of an ontic action execution is $2n$ where only the updated possibilities (n) are new elements of $\mathcal{S}$.

- The case with *sensing* and *un-trustworthy announcement* actions is similar to the ontic action one.
- Finally, *mis-trustworthy announcement* generates up to $2n$ new possibilities. Each possibility $\in |\mathcal{B}^{\mathsf{p}}_{\mathcal{AG}}|$ can be updated with the announced value—if reached by a trusty fully observant agent—or updated with the negation of the announced fluent—if reached by an untrusty fully observant agent. Both of the copies can then be added to the unchanged possibilities—reached by an oblivious agent—meaning that the size of $\mathcal{B}^{\mathsf{p}'}_{\mathcal{AG}}$ in case of an *mis*-trustworthy announcement action execution is $3n$ where only the updated possibilities ($2n$) are new elements of $\mathcal{S}$.

This identifies $3n$ as the upper bound for the growth of a state size and for the generation of new possibilities after an action execution. Therefore, given the size n of the initial state and the length of the action sequence l we can conclude that $|\mathcal{S}| \leq (n \times 3^l)$ that is indeed finite. □

A.3.2 Proofs

Preserving all the other concepts introduced in Appendix A.1, we are ready to prove Proposition 3.1.

Proof of Proposition 3.1 Let us prove each item separately:

(1) In the following we prove the first property of Proposition 3.1.

- First of all we identify the set of all the possibilities reached by the *trusty fully observant* agents in u as $\mathcal{B}^{\mathsf{u}}_{\mathbf{F}_\mathsf{a}}$ and we recall that, as shown in Appendix A.1, this set corresponds to the possibilities reached by $\mathbf{C}_{\mathbf{F}_\mathsf{a}}$.
- We then calculate $\mathcal{B}^{\mathsf{u}'}_{\mathbf{F}_\mathsf{a}}$ that, following Definition 3.2, contains only possibilities p' such that $e(\mathsf{a}, \mathsf{p}') = 0$ (False).
- This means that $\forall \mathsf{p}' \in \mathcal{B}^{\mathsf{u}'}_{\mathbf{F}_\mathsf{a}}$ we have that $\mathsf{p}' \models \phi$.
- Given that this set contains only possibilities that entail ϕ we can derive that $\mathcal{B}^{\mathsf{u}'}_{\mathbf{F}_\mathsf{a}} \models \phi$.
- Finally, as the set $\mathbf{C}_{\mathbf{F}_\mathsf{a}} \equiv \mathcal{B}^{\mathsf{u}'}_{\mathbf{F}_\mathsf{a}}$, we have that $\mathbf{C}_{\mathbf{F}_\mathsf{a}} \models \phi$.

(2) The set of the possibilities reachable by *untrusty fully observant* agents is $\mathcal{B}^{\mathsf{u}}_{\mathbf{U}_\mathsf{a}}$.

- In order to compute $\mathcal{B}^{\mathsf{u}'}_{\mathbf{U}_\mathsf{a}}$, following Definition 3.2, we apply $\Psi(\mathsf{a},\mathsf{u})$ to every element of $\mathcal{B}^{\mathsf{u}}_{\mathbf{U}_\mathsf{a}}$.

- This means that the set of beliefs of the trusty fully observant, from the point of view of the untrusty ones, is represented by the set $\mathcal{B}^{\mathsf{u}'}_{\mathbf{U}_\mathsf{a},\mathbf{F}_\mathsf{a}} = \{\mathsf{p}' \mid \mathsf{p} \in \mathcal{B}^{\mathsf{u}'}_{\mathbf{U}_\mathsf{a}} \wedge e(\mathsf{a},\mathsf{p}') = 0\}$.

- We then have that $\forall p' \in \mathcal{B}^{\mathsf{u}'}_{\mathbf{U}_\mathsf{a},\mathbf{F}_\mathsf{a}}\ p' \models \phi$ and therefore that $\mathsf{u}' \models \mathbf{C}_{\mathbf{U}_\mathsf{a}}(\mathbf{C}_{\mathbf{F}_\mathsf{a}}\phi)$.

(3) Let us identify the set of the possibilities reachable by *partial observant* agents with $\mathcal{B}^{\mathsf{u}}_{\mathbf{P}_\mathsf{a}}$. We also recall that this set is equal to $\mathbf{C}_{\mathbf{P}_\mathsf{a}}$ in u.

- Now to calculate $\mathcal{B}^{\mathsf{u}'}_{\mathbf{P}_\mathsf{a}}$, following Definition 3.2, we apply $\Upsilon(\mathsf{a},\mathsf{w}) \cup \Psi(\mathsf{a},\mathsf{u})$ to every element of $\mathcal{B}^{\mathsf{u}}_{\mathbf{P}_\mathsf{a}}$.

- It is easy to identify two disjoint subsets $\mathcal{B}^{\Upsilon}_{\mathbf{P}_\mathsf{a}}$ and $\mathcal{B}^{\Psi}_{\mathbf{P}_\mathsf{a}}$ of $\mathcal{B}^{\mathsf{u}'}_{\mathbf{P}_\mathsf{a}}$ that contain only possibilities such that:
  - $\mathcal{B}^{\Psi}_{\mathbf{P}_\mathsf{a}} \models \mathsf{u}(\mathsf{a},\mathsf{u})$;
  - $\mathcal{B}^{\Upsilon}_{\mathbf{P}_\mathsf{a}} \not\models \mathsf{u}(\mathsf{a},\mathsf{u})$;
  - $(\mathcal{B}^{\Psi}_{\mathbf{P}_\mathsf{a}} \cup \mathcal{B}^{\Upsilon}_{\mathbf{P}_\mathsf{a}}) \equiv \mathcal{B}^{\mathsf{u}'}_{\mathbf{P}_\mathsf{a}}$; and
  - $(\mathcal{B}^{\Psi}_{\mathbf{P}_\mathsf{a}} \cap \mathcal{B}^{\Upsilon}_{\mathbf{P}_\mathsf{a}}) \equiv \emptyset$.

- From these two sets we can now construct the sets $\mathcal{B}^{\Psi}_{\mathbf{P}_\mathsf{a},\mathbf{F}_\mathsf{a}}$ and $\mathcal{B}^{\Upsilon}_{\mathbf{P}_\mathsf{a},\mathbf{F}_\mathsf{a}}$ that are simply the set of possibilities reachable from the *fully observant* agents starting from $\mathcal{B}^{\Psi}_{\mathbf{P}_\mathsf{a}}$ and $\mathcal{B}^{\Upsilon}_{\mathbf{P}_\mathsf{a}}$, respectively.

- Given that the set $\mathcal{B}^{\Psi}_{\mathbf{P}_\mathsf{a},\mathbf{F}_\mathsf{a}}$ resulted from the application of the transition function from the point of view of trusty fully observant agents, we know from Item **(1)** that for $\forall \mathsf{p} \in \mathcal{B}^{\Psi}_{\mathbf{P}_\mathsf{a},\mathbf{F}_\mathsf{a}}, \mathsf{p} \models \phi$.

- This implies that $\mathcal{B}^{\Psi}_{\mathbf{P}_\mathsf{a},\mathbf{F}_\mathsf{a}}$ reaches only possibilities where the interpretation of ϕ is true and similarly in $\mathcal{B}^{\Upsilon}_{\mathbf{P}_\mathsf{a},\mathbf{F}_\mathsf{a}}$ only possibilities where the interpretation of ϕ is false.

- This means that $\mathcal{B}^{\Psi}_{\mathbf{P}_\mathsf{a},\mathbf{F}_\mathsf{a}} \models \phi$ and $\mathcal{B}^{\Upsilon}_{\mathbf{P}_\mathsf{a},\mathbf{F}_\mathsf{a}} \models \neg\phi$.

- We can then derive that $\mathcal{B}^{\Psi}_{\mathbf{P}_\mathsf{a}} \models \mathbf{C}_{\mathbf{F}_\mathsf{a}}\phi$ being $\mathcal{B}^{\Psi}_{\mathbf{P}_\mathsf{a},\mathbf{F}_\mathsf{a}} = \{\mathsf{p} \mid \mathsf{p} \in \bigcup_{\mathsf{q} \in \mathcal{B}^{\Psi}_{\mathbf{P}_\mathsf{a}}} \mathsf{q}(\mathbf{F}_\mathsf{a})\}$ (and similarly $\mathcal{B}^{\Upsilon}_{\mathbf{P}_\mathsf{a}} \models \mathbf{C}_{\mathbf{F}_\mathsf{a}}\neg\phi$).

- Finally, being $\mathcal{B}^{\mathsf{u}'}_{\mathbf{P}_\mathsf{a}} = \mathcal{B}^{\Psi}_{\mathbf{P}_\mathsf{a}} \cup \mathcal{B}^{\Upsilon}_{\mathbf{P}_\mathsf{a}}$ we can conclude that $\mathcal{B}^{\mathsf{u}'}_{\mathbf{P}_\mathsf{a}} \models \mathbf{C}_{\mathbf{F}_\mathsf{a}}\phi \vee \mathbf{C}_{\mathbf{F}_\mathsf{a}}\neg\phi^a$ and therefore $\mathsf{u}' \models \mathbf{C}_{\mathbf{P}_\mathsf{a}}(\mathbf{C}_{\mathbf{F}_\mathsf{a}}\phi \vee \mathbf{C}_{\mathbf{F}_\mathsf{a}}\neg\phi)$.

(4) To prove this item we will make use of the properties proved in previous Items.

- As said in Appendix A.1, we know that $\mathcal{B}^{\mathsf{u}}_{\mathbf{F}_\mathsf{a}\cup\mathbf{U}_\mathsf{a},\mathbf{P}_\mathsf{a}}$ corresponds with the set of possibilities identified by $\mathbf{C}_{\mathbf{F}_\mathsf{a}\cup\mathbf{U}_\mathsf{a}}\mathbf{C}_{\mathbf{P}_\mathsf{a}}$ and it is also equal to $\{\mathsf{p} \mid (\exists \mathsf{q} \in \mathcal{B}^{\mathsf{u}}_{\mathbf{P}_\mathsf{a}})(\mathsf{p} \in \bigcup_{\mathsf{i}\in\mathbf{F}_\mathsf{a}\cup\mathbf{U}_\mathsf{a}} \mathsf{q}(\mathsf{i}))\}$.
- Now to calculate $\mathcal{B}^{\mathsf{u}'}_{\mathbf{F}_\mathsf{a}\cup\mathbf{U}_\mathsf{a}}$ we apply Definition 3.2 to every element of $\mathcal{B}^{\mathsf{u}}_{\mathbf{F}_\mathsf{a}\cup\mathbf{U}_\mathsf{a}}$. This means that $\mathcal{B}^{\mathsf{u}'}_{\mathbf{F}_\mathsf{a}\cup\mathbf{U}_\mathsf{a}} = \{\mathsf{p}' \mid (\exists \mathsf{p} \in \mathcal{B}^{\mathsf{u}}_{\mathbf{F}_\mathsf{a}\cup\mathbf{U}_\mathsf{a}})(\mathsf{p}' = \Psi(\mathsf{a}, \mathsf{p}))\}$.
- We then want to calculate the set $\{\mathsf{p}' \mid (\exists \mathsf{q}' \in \mathcal{B}^{\mathsf{u}'}_{\mathbf{F}_\mathsf{a}\cup\mathbf{U}_\mathsf{a}})(\mathsf{p}' \in \bigcup_{\mathsf{i}\in\mathbf{P}_\mathsf{a}} \mathsf{q}'(\mathsf{i}))\}$.
- To calculate the "point of view" of the partially observants with respect to the fully observants we apply Definition 3.2 to all the elements of $\{\mathsf{p} \mid (\exists \mathsf{q}' \in \mathcal{B}^{\mathsf{u}'}_{\mathbf{F}_\mathsf{a}\cup\mathbf{U}_\mathsf{a}})(\mathsf{p} \in \mathcal{B}^{\mathsf{q}}_{\mathbf{P}_\mathsf{a}})\}$.
- We can then derive that the resulting set is $\{\mathsf{p}' \mid (\exists \mathsf{q}' \in \mathcal{B}^{\mathsf{u}'}_{\mathbf{F}_\mathsf{a}\cup\mathbf{U}_\mathsf{a}})(\mathsf{p}' \in \bigcup_{\mathsf{i}\in\mathbf{P}_\mathsf{a}} \mathsf{q}'(\mathsf{i}))\} \equiv \mathcal{B}^{\mathsf{u}'}_{\mathbf{F}_\mathsf{a}\cup\mathbf{U}_\mathsf{a},\mathbf{P}_\mathsf{a}}$.
- We showed in the previous item that given the set of possibilities resulted by applying the transition function entails $\mathbf{C}_{\mathbf{F}_\mathsf{a}\cup\mathbf{U}_\mathsf{a}}\phi \vee \mathbf{C}_{\mathbf{F}_\mathsf{a}\cup\mathbf{U}_\mathsf{a}}\neg\phi$.
- This means that $\mathcal{B}^{\mathsf{u}'}_{\mathbf{F}_\mathsf{a}\cup\mathbf{U}_\mathsf{a},\mathbf{P}_\mathsf{a}} \models (\mathbf{C}_{\mathbf{F}_\mathsf{a}\cup\mathbf{U}_\mathsf{a}}\phi \vee \mathbf{C}_{\mathbf{F}_\mathsf{a}\cup\mathbf{U}_\mathsf{a}}\neg\phi)$ and therefore, following what said in Appendix A.1, $\mathsf{u}' \models \mathbf{C}_{\mathbf{F}_\mathsf{a}\cup\mathbf{U}_\mathsf{a}}(\mathbf{C}_{\mathbf{P}_\mathsf{a}}(\mathbf{C}_{\mathbf{F}_\mathsf{a}\cup\mathbf{U}_\mathsf{a}}\phi \vee \mathbf{C}_{\mathbf{F}_\mathsf{a}\cup\mathbf{U}_\mathsf{a}}\neg\phi))$.

(5) Let us consider $\mathsf{y} \in \mathbf{O}_\mathsf{a}$.

- If $\mathsf{u} \models \mathbf{B}_\mathsf{y}(\varphi)$ we have that every $\mathsf{p} \in \mathcal{B}^{\mathsf{u}}_\mathsf{y}$ entails φ.
- Given that, from Definition 3.2, when $\mathsf{y} \in \mathbf{O}_\mathsf{a}$ for each possibility $\mathsf{p} \in \mathcal{B}^{\mathsf{u}}_\mathsf{y}$ $\mathsf{p}(\mathsf{y}) = \mathsf{p}'(\mathsf{y})$ it is easy to see that $\mathcal{B}^{\mathsf{u}}_\mathsf{y} \equiv \mathcal{B}^{\mathsf{u}'}_\mathsf{y}$.
- Given that the two sets of possibilities are the same it means that the reachability functions that they represent are the same.
- Being the two functions the same it means that $\forall\varphi \in D$ $\mathsf{u} \models \mathbf{B}_\mathsf{y}(\varphi)$ *iff* $\mathsf{u}' \models \mathbf{B}_\mathsf{y}(\varphi)$.

(6) Let us consider $\mathsf{x} \in \mathbf{F}_\mathsf{a} \cup \mathbf{U}_\mathsf{a} \cup \mathbf{P}_\mathsf{a}$ and $\mathsf{y} \in \mathbf{O}_\mathsf{a}$.

- Being $\mathsf{y} \in \mathbf{O}_\mathsf{a}$, from Definition 3.2, we know that $\mathsf{p}(\mathsf{y}) = \mathsf{p}'(\mathsf{y})$ such that $\mathsf{p} \in \mathcal{B}^\mathsf{u}_\mathsf{x}$ and p' is its updated version $\in \mathcal{B}^{\mathsf{u}'}_\mathsf{x}$.
- This means that for every element in $\mathcal{B}^\mathsf{u}_\mathsf{x}$ we have an updated version that has the same reachability function for the agent y.
- Then we can derive that $\mathcal{B}^\mathsf{u}_{\mathsf{x},\mathsf{y}} \equiv \mathcal{B}^{\mathsf{u}'}_{\mathsf{x},\mathsf{y}}$ and therefore that these two sets contain the same possibilities.
- As already said in Item **(5)** when two sets of possibilities are the same they entail the same formulae.
- We can conclude that if $\mathsf{u} \models \mathbf{B}_\mathsf{x}(\mathbf{B}_\mathsf{y}(\varphi))$ then $\mathsf{u}' \models \mathbf{B}_\mathsf{x}(\mathbf{B}_\mathsf{y}(\varphi))$.

(7) Let us consider $\mathsf{y} \in \mathbf{U}_\mathsf{a}$. Let us assume, without losing generality, that $\mathsf{u} \models \mathbf{B}_\mathsf{y}(\phi)$. The cases where $\mathsf{u} \models \mathbf{B}_\mathsf{y}(\neg\phi)$ or $\mathsf{u} \models (\neg\mathbf{B}_\mathsf{y}(\phi) \wedge \neg\mathbf{B}_\mathsf{y}(\neg\phi))$ are similar and therefore omitted.

- Being $\mathsf{y} \in \mathbf{U}_\mathsf{a}$ we know that the updated version of her/his reachable possibility is $\mathcal{B}^{\mathsf{u}'}_\mathsf{y} = \{\mathsf{p}' \mid \mathsf{p} \in \mathcal{B}^\mathsf{u}_\mathsf{y} \wedge \mathsf{p}' = \Psi(\mathsf{a}, \mathsf{p})\}$.
- Following Definition 3.2 we know that each possibility in $\mathcal{B}^{\mathsf{u}'}_\mathsf{y}$ has the same fluent set of its previous version.
- Moreover, an untrusty agent preserves all the edges. This means that if an agent reached q from q in u she/he will reach q' from q' in u'.
- From the last statement, and given that the updated version of each possibility maintains the same fluent set we can conclude that, if $\mathsf{u} \models \mathbf{B}_\mathsf{y}(\phi)$ iff $\mathsf{u} \models \mathbf{B}_\mathsf{y}(\phi)$ (similarly if $\mathsf{u} \models \mathbf{B}_\mathsf{y}(\neg\phi)$ and if $(\neg\mathbf{B}_\mathsf{y}(\phi) \wedge \neg\mathbf{B}_\mathsf{y}(\neg\phi))$).

□

[a] The two sets are completely disjoint as one only contains possibilities that entail ϕ while the other only possibilities that do not. This means that that does not exist any fully-observant-edge between possibilities that belongs in two different sets.

Finally, we can prove the properties introduced in Proposition 3.2.

Proof of Proposition 3.2 Let us prove each property separately.

(8) In the following we prove the first property of Proposition 3.2.

- First of all we identify the set of all the possibilities reached by the

untrusty fully observant agents in u as $\mathcal{B}^{\mathsf{u}}_{\mathbf{U}_\mathsf{a}}$ and we recall that, as shown in Appendix A.1, this set corresponds to the possibilities reached by $\mathbf{C}_{\mathbf{U}_\mathsf{a}}$.

- We then calculate $\mathcal{B}^{\mathsf{u}'}_{\mathbf{U}_\mathsf{a}}$ that, following Definition 3.3, contains only possibilities p' such that $e(\mathsf{a}, \mathsf{p}') = 1$ (True).
- This means that $\forall \mathsf{p}' \in \mathcal{B}^{\mathsf{u}'}_{\mathbf{F}_\mathsf{a}}$ we have that $\mathsf{p}' \models \neg\phi$.
- Given that this set contains only possibilities that entail ϕ we can derive that $\mathcal{B}^{\mathsf{u}'}_{\mathbf{U}_\mathsf{a}} \models \neg\phi$.
- Finally, as the set $\mathbf{C}_{\mathbf{U}_\mathsf{a}} \equiv \mathcal{B}^{\mathsf{u}'}_{\mathbf{U}_\mathsf{a}}$, we have that $\mathbf{C}_{\mathbf{U}_\mathsf{a}} \models \neg\phi$.

(9) The set of the possibilities reachable by *trusty fully observant* agents is $\mathcal{B}^{\mathsf{u}}_{\mathbf{F}_\mathsf{a}}$.

- Now to calculate $\mathcal{B}^{\mathsf{u}'}_{\mathbf{F}_\mathsf{a}}$, following Definition 3.3, we apply $\Psi(\mathsf{a}, \mathsf{u})$ to every element of $\mathcal{B}^{\mathsf{u}}_{\mathbf{F}_\mathsf{a}}$.
- This means that the set of beliefs of the untrusty fully observant, from the point of view of the trusty ones, is represented by the set $\mathcal{B}^{\mathsf{u}'}_{\mathbf{F}_\mathsf{a},\mathbf{U}_\mathsf{a}} = \{\mathsf{p}' \mid \mathsf{p} \in \mathcal{B}^{\mathsf{u}'}_{\mathbf{F}_\mathsf{a}} \wedge e(\mathsf{a}, \mathsf{p}') = 1\}$.
- It is then straightforward to see that the $\forall p' \in \mathcal{B}^{\mathsf{u}'}_{\mathbf{F}_\mathsf{a},\mathbf{U}_\mathsf{a}}\ p' \models \neg\phi$ and therefore that $\mathsf{u}' \models \mathbf{C}_{\mathbf{F}_\mathsf{a}}(\mathbf{C}_{\mathbf{U}_\mathsf{a}}\neg\phi)$.

(10) We identify the set of the possibilities reachable by *partial observant* agents with $\mathcal{B}^{\mathsf{u}}_{\mathbf{P}_\mathsf{a}}$. We also recall that this set is equal to $\mathbf{C}_{\mathbf{P}_\mathsf{a}}$ in u.

- Now to calculate $\mathcal{B}^{\mathsf{u}'}_{\mathbf{P}_\mathsf{a}}$, following Definition 3.3, we apply $\Upsilon(\mathsf{a}, \mathsf{w}) \cup \Psi(\mathsf{a}, \mathsf{u})$ to every element of $\mathcal{B}^{\mathsf{u}}_{\mathbf{P}_\mathsf{a}}$.
- It is easy to identify two disjoint subsets $\mathcal{B}^{\Upsilon}_{\mathbf{P}_\mathsf{a}}$ and $\mathcal{B}^{\Psi}_{\mathbf{P}_\mathsf{a}}$ of $\mathcal{B}^{\mathsf{u}'}_{\mathbf{P}_\mathsf{a}}$ that contains only possibility such that:
  - $\mathcal{B}^{\Psi}_{\mathbf{P}_\mathsf{a}} \models \mathsf{u}(\mathsf{a}, \mathsf{u})$;
  - $\mathcal{B}^{\Upsilon}_{\mathbf{P}_\mathsf{a}} \not\models \mathsf{u}(\mathsf{a}, \mathsf{u})$;
  - $(\mathcal{B}^{\Psi}_{\mathbf{P}_\mathsf{a}} \cup \mathcal{B}^{\Upsilon}_{\mathbf{P}_\mathsf{a}}) \equiv \mathcal{B}^{\mathsf{u}'}_{\mathbf{P}_\mathsf{a}}$; and
  - $(\mathcal{B}^{\Psi}_{\mathbf{P}_\mathsf{a}} \cap \mathcal{B}^{\Upsilon}_{\mathbf{P}_\mathsf{a}}) \equiv \emptyset$.
- From these two sets we can now construct the sets $\mathcal{B}^{\Psi}_{\mathbf{P}_\mathsf{a},\mathbf{U}_\mathsf{a}}$ and $\mathcal{B}^{\Upsilon}_{\mathbf{P}_\mathsf{a},\mathbf{U}_\mathsf{a}}$

that are simply the set of possibilities reachable from the *fully observant* agents starting from $\mathcal{B}^{\Psi}_{\mathbf{P}_a}$ and $\mathcal{B}^{\Upsilon}_{\mathbf{P}_a}$ respectively.

- Given that the set $\mathcal{B}^{\Psi}_{\mathbf{P}_a,\mathbf{U}_a}$ resulted from the application of the transition function from the point of view of untrusty fully observant agents, we know from Item **(8)** that for $\forall \mathsf{p} \in \mathcal{B}^{\Psi}_{\mathbf{P}_a,\mathbf{U}_a}, \mathsf{p} \models \neg\phi$.

- This implies that $\mathcal{B}^{\Psi}_{\mathbf{P}_a,\mathbf{U}_a}$ reaches only possibilities where the interpretation of ϕ is false and similarly in $\mathcal{B}^{\Upsilon}_{\mathbf{P}_a,\mathbf{U}_a}$ only possibilities where the interpretation of ϕ is true.

- This means that $\mathcal{B}^{\Psi}_{\mathbf{P}_a,\mathbf{U}_a} \models \neg\phi$ and $\mathcal{B}^{\Upsilon}_{\mathbf{P}_a,\mathbf{U}_a} \models \phi$.

- We can then derive that $\mathcal{B}^{\Psi}_{\mathbf{P}_a} \models \mathbf{C}_{\mathbf{U}_a}\neg\phi$ being $\mathcal{B}^{\Psi}_{\mathbf{P}_a,\mathbf{U}_a} = \{\mathsf{p} \mid \mathsf{p} \in \bigcup_{\mathsf{q}\in\mathcal{B}^{\Psi}_{\mathbf{P}_a}} \mathsf{q}(\mathbf{U}_a)\}$ (and similarly $\mathcal{B}^{\Upsilon}_{\mathbf{P}_a} \models \mathbf{C}_{\mathbf{U}_a}\phi$).

- Finally, being $\mathcal{B}^{\mathsf{u}'}_{\mathbf{P}_a} = \mathcal{B}^{\Psi}_{\mathbf{P}_a} \cup \mathcal{B}^{\Upsilon}_{\mathbf{P}_a}$ we can conclude that $\mathcal{B}^{\mathsf{u}'}_{\mathbf{P}_a} \models \mathbf{C}_{\mathbf{U}_a}\phi \vee \mathbf{C}_{\mathbf{U}_a}\neg\phi$ and therefore $\mathsf{u}' \models \mathbf{C}_{\mathbf{P}_a}(\mathbf{C}_{\mathbf{U}_a}\phi \vee \mathbf{C}_{\mathbf{U}_a}\neg\phi)$.

□

A.4 Proof of Proposition 4.1

Following we will present the formal proof that the properties presented in Proposition 4.1.

Proof of Proposition 4.1 Let us prove each item separately. Let us assume that $\mathtt{a}$ is $\mathtt{j}$ **announces** $\mathtt{f}$. The case when $\mathtt{j}$ **announces** $\neg\mathtt{f}$ is similar, and we will only highlight the differences when it is needed.

(1) In the following we prove Item **(1)**.

- First of all we identify the set of all the possibilities reached by the *fully observant* agents in $\mathtt{u}$ as $\mathcal{B}^{\mathtt{u}}_{\mathbf{F}_\mathtt{a}}$.
- We then re-apply the reachability function following the beliefs of the `trustful` agents. This means that the set of beliefs of the `trustful`, from the point of view of the *fully observant* ones, is represented by the set $\mathcal{B}^{\mathtt{u}}_{\mathbf{F}_\mathtt{a},\mathbf{T}_\mathtt{a}} = \{\mathtt{p} \mid \mathtt{p} \in \mathcal{B}^{\mathtt{q}}_{\mathbf{T}_\mathtt{a}} \wedge \mathtt{q} \in \mathcal{B}^{\mathtt{u}}_{\mathbf{F}_\mathtt{a}}\}$.
- Now to calculate $\mathcal{B}^{\mathtt{u}'}_{\mathbf{F}_\mathtt{a},\mathbf{T}_\mathtt{a}}$, following Definition 4.4, we apply $\chi(\mathtt{f},\mathtt{p},1)$ to every element $\mathtt{p}$ of $\mathcal{B}^{\mathtt{u}}_{\mathbf{F}_\mathtt{a},\mathbf{T}_\mathtt{a}}$. Let us note that if $e(\mathtt{a}) = 0$, that is if $\mathtt{j}$ **announces** $\neg\mathtt{f}$, we should apply $\chi(\mathtt{f},\mathtt{p},0)$ instead.
- This means that the set of updated beliefs of `trustful` agents, from the point of view of the *fully observant* ones, is represented by the set $\mathcal{B}^{\mathtt{u}'}_{\mathbf{F}_\mathtt{a},\mathbf{T}_\mathtt{a}} = \{\mathtt{p}' \mid \mathtt{p}'(\mathcal{F}) = ((\mathtt{p}(\mathcal{F}) \setminus \{\neg\mathtt{f}\}) \cup \{\mathtt{f}\}) \wedge \mathtt{p} \in \mathcal{B}^{\mathtt{u}}_{\mathbf{F}_\mathtt{a},\mathbf{T}_\mathtt{a}}\}$. It is important to notice that the truth value of the fluent $\mathtt{f}$ in the set of possibilities $\mathcal{B}^{\mathtt{u}'}_{\mathbf{F}_\mathtt{a}}$ is not important as the application of $\chi(\mathtt{f},\mathtt{p},1)$ on all these possibilities forces their updated version to set the truth value of $\mathtt{f} = 1$ (similarly, for the negated case, the fluent truth value is 0).
- It is then straightforward to see that the set $\mathcal{B}^{\mathtt{u}'}_{\mathbf{F}_\mathtt{a},\mathbf{T}_\mathtt{a}}$ entails $\mathtt{f}$, as all the reached possibility have the truth value of $\mathtt{f}$ set to 1. Recalling that, as shown in Appendix A.1, the set $\mathcal{B}^{\mathtt{u}}_{\alpha,\beta}$ corresponds to the possibilities reached by $\mathbf{C}_\alpha(\mathbf{C}_\beta)$ where $\alpha,\beta \subseteq D(\mathcal{AG})$ it is clear that the updated e-state $\mathtt{u}' \models \mathbf{C}_{\mathbf{F}_\mathtt{a}}(\mathbf{C}_{\mathbf{T}_\mathtt{a}}(\mathtt{f}))$ (and similarly, in the negated case, $\mathtt{u}' \models \mathbf{C}_{\mathbf{F}_\mathtt{a}}(\mathbf{C}_{\mathbf{T}_\mathtt{a}}(\neg\mathtt{f}))$).
- Now, to show that $\mathtt{u}' \models \mathbf{C}_{\mathbf{F}_\mathtt{a}}(\mathbf{C}_{\mathbf{T}_\mathtt{a}}(\mathbf{B}_\mathtt{j}(\mathtt{f})))$ we need to recall that the `trustful` agents consider that the announcer $\mathtt{j}$ to be `trustful` as well. This means that $\mathcal{B}^{\mathtt{u}'}_{\mathbf{F}_\mathtt{a},\mathbf{T}_\mathtt{a},\{\mathtt{j}\}}$ is equal to $\mathcal{B}^{\mathtt{u}'}_{\mathbf{F}_\mathtt{a},\mathbf{T}_\mathtt{a}}$ as the `trustful` agents believes that the announcer $\mathtt{j}$ used $\chi(\mathtt{f},\mathtt{p},1)$ to update her/his beliefs (being, from their perspective `trustful` agent). This means that all the possibilities in $\mathcal{B}^{\mathtt{u}'}_{\mathbf{F}_\mathtt{a},\mathbf{T}_\mathtt{a},\{\mathtt{j}\}}$ have the truth value of $\mathtt{f}$ set to 1 (or 0 in

the negated case).

- Following Appendix A.1 we know that $\mathcal{B}^{\mathsf{u}'}_{\mathbf{F}_\mathsf{a},\mathbf{T}_\mathsf{a},\{\mathsf{j}\}}$ is equal to the possibilities reached by applying $\mathbf{C}_{\mathbf{F}_\mathsf{a}}(\mathbf{C}_{\mathbf{T}_\mathsf{a}}(\mathbf{B}_\mathsf{j}))$. Given that all these possibilities have the truth value of $\mathtt{f}$ set to 1 it is straightforward to see that $\mathsf{u}' \models \mathbf{C}_{\mathbf{F}_\mathsf{a}}(\mathbf{C}_{\mathbf{T}_\mathsf{a}}(\mathbf{B}_\mathsf{j}(\mathtt{f})))$.

- From the previous items now know that $\mathsf{u}' \models \mathbf{C}_{\mathbf{F}_\mathsf{a}}(\mathbf{C}_{\mathbf{T}_\mathsf{a}}(\mathtt{f})) \wedge \mathbf{C}_{\mathbf{F}_\mathsf{a}}(\mathbf{C}_{\mathbf{T}_\mathsf{a}}(\mathbf{B}_\mathsf{j}(\mathtt{f})))$ and therefore that $\mathsf{u}' \models \mathbf{C}_{\mathbf{F}_\mathsf{a}}(\mathbf{C}_{\mathbf{T}_\mathsf{a}}(\mathtt{f} \wedge \mathbf{B}_\mathsf{j}(\mathtt{f})))$ as stated in Item **(1)** (while for the negated case we can easily derive that $\mathsf{u}' \models \mathbf{C}_{\mathbf{F}_\mathsf{a}}(\mathbf{C}_{\mathbf{T}_\mathsf{a}}(\neg\mathtt{f} \wedge \mathbf{B}_\mathsf{j}(\neg\mathtt{f}))))$.

(2) Let us proceed with Item **(2)**.

- First we identify the set of all the possibilities reached by the *fully observant* agents in u as $\mathcal{B}^{\mathsf{u}}_{\mathbf{F}_\mathsf{a}}$.

- We then re-apply the reachability function following the beliefs of the `mistrustful` agents. This means that the set of beliefs of the `mistrustful`, from the point of view of the *fully observant* ones, is represented by the set $\mathcal{B}^{\mathsf{u}}_{\mathbf{F}_\mathsf{a},\mathbf{M}_\mathsf{a}} = \{\mathsf{p} \mid \mathsf{p} \in \mathcal{B}^{\mathsf{q}}_{\mathbf{M}_\mathsf{a}} \wedge \mathsf{q} \in \mathcal{B}^{\mathsf{u}}_{\mathbf{F}_\mathsf{a}}\}$.

- Now to calculate $\mathcal{B}^{\mathsf{u}'}_{\mathbf{F}_\mathsf{a},\mathbf{M}_\mathsf{a}}$, following Definition 4.4, we apply $\chi(\mathtt{f},\mathsf{p},0)$ to every element p of $\mathcal{B}^{\mathsf{u}}_{\mathbf{F}_\mathsf{a},\mathbf{M}_\mathsf{a}}$. Let us note that if $e(\mathsf{a}) = 0$, that is if j **announces** $\neg\mathtt{f}$, we should apply $\chi(\mathtt{f},\mathsf{p},1)$ instead.

- This means that the set of updated beliefs of `mistrustful` agents, from the point of view of the *fully observant* ones, is represented by the set $\mathcal{B}^{\mathsf{u}'}_{\mathbf{F}_\mathsf{a},\mathbf{M}_\mathsf{a}} = \{\mathsf{p}' \mid \mathsf{p}'(\mathcal{F}) = ((\mathsf{p}(\mathcal{F}) \setminus \{\mathtt{f}\}) \cup \{\neg\mathtt{f}\}) \wedge \mathsf{p} \in \mathcal{B}^{\mathsf{u}}_{\mathbf{F}_\mathsf{a},\mathbf{M}_\mathsf{a}}\}$. It is important to notice that the truth value of the fluent $\mathtt{f}$ in the set of possibilities $\mathcal{B}^{\mathsf{u}'}_{\mathbf{F}_\mathsf{a}}$ is not important as the application of $\chi(\mathtt{f},\mathsf{p},0)$ on all these possibilities forces their updated version to set the truth value of $\mathtt{f} = 0$ (similarly, for the negated case, the fluent truth value is 1).

- It is then straightforward to see that the set $\mathcal{B}^{\mathsf{u}'}_{\mathbf{F}_\mathsf{a},\mathbf{M}_\mathsf{a}}$ entails $\neg\mathtt{f}$, as all the reached possibility have the truth value of $\mathtt{f}$ set to 0. Recalling that, as shown in Appendix A.1, the set $\mathcal{B}^{\mathsf{u}}_{\alpha,\beta}$ corresponds to the possibilities reached by $\mathbf{C}_\alpha(\mathbf{C}_\beta)$ where $\alpha, \beta \subseteq D(\mathcal{AG})$ it is clear that the updated e-state $\mathsf{u}' \models \mathbf{C}_{\mathbf{F}_\mathsf{a}}(\mathbf{C}_{\mathbf{M}_\mathsf{a}}(\neg\mathtt{f}))$ (and similarly, in the negated case, $\mathsf{u}' \models \mathbf{C}_{\mathbf{F}_\mathsf{a}}(\mathbf{C}_{\mathbf{M}_\mathsf{a}}(\mathtt{f})))$.

- Now, to prove that $\mathsf{u}' \models \mathbf{C}_{\mathbf{F}_\mathsf{a}}(\mathbf{C}_{\mathbf{M}_\mathsf{a}}(\mathbf{B}_\mathsf{j}(\neg\mathtt{f})))$ we need to recall that the `mistrustful` agents consider that the announcer j to be

mistrustful as well. This means that $\mathcal{B}^{\mathsf{u}'}_{\mathbf{F}_\mathsf{a},\mathbf{M}_\mathsf{a},\{\mathsf{j}\}}$ is equal to $\mathcal{B}^{\mathsf{u}'}_{\mathbf{F}_\mathsf{a},\mathbf{M}_\mathsf{a}}$ as the mistrustful agents believes that the announcer j used $\chi(\mathtt{f},\mathsf{p},0)$ to update her/his beliefs (being, from their perspective mistrustful agent). This means that all the possibilities in $\mathcal{B}^{\mathsf{u}'}_{\mathbf{F}_\mathsf{a},\mathbf{M}_\mathsf{a},\{\mathsf{j}\}}$ have the truth value of f set to 0 (or 1 in the negated case).

- Following Appendix A.1 we know that $\mathcal{B}^{\mathsf{u}'}_{\mathbf{F}_\mathsf{a},\mathbf{M}_\mathsf{a},\{\mathsf{j}\}}$ is equal to the possibilities reached by applying $\mathbf{C}_{\mathbf{F}_\mathsf{a}}(\mathbf{C}_{\mathbf{M}_\mathsf{a}}(\mathbf{B}_\mathsf{j}))$. Given that all these possibilities have the truth value of f set to 0 it is straightforward to see that $\mathsf{u}' \models \mathbf{C}_{\mathbf{F}_\mathsf{a}}(\mathbf{C}_{\mathbf{M}_\mathsf{a}}(\mathbf{B}_\mathsf{j}(\neg\mathtt{f})))$.

- From the previous items now know that $\mathsf{u}' \models \mathbf{C}_{\mathbf{F}_\mathsf{a}}(\mathbf{C}_{\mathbf{M}_\mathsf{a}}(\neg\mathtt{f})) \wedge \mathbf{C}_{\mathbf{F}_\mathsf{a}}(\mathbf{C}_{\mathbf{M}_\mathsf{a}}(\mathbf{B}_\mathsf{j}(\neg\mathtt{f})))$ and therefore that $\mathsf{u}' \models \mathbf{C}_{\mathbf{F}_\mathsf{a}}(\mathbf{C}_{\mathbf{M}_\mathsf{a}}(\neg\mathtt{f} \wedge \mathbf{B}_\mathsf{j}(\neg\mathtt{f})))$ as stated in Item **(1)** (while for the negated case we can easily derive that $\mathsf{u}' \models \mathbf{C}_{\mathbf{F}_\mathsf{a}}(\mathbf{C}_{\mathbf{M}_\mathsf{a}}(\mathtt{f} \wedge \mathbf{B}_\mathsf{j}(\mathtt{f}))))$.

(3) To prove Item **(3)** let us consider $\mathsf{i} \in (\mathbf{S}_\mathsf{a} \cup \{\mathsf{j}\})$ and that u does entail φ (where $\varphi \in \{\mathbf{B}_\mathsf{i}(\ell),\ \mathbf{B}_\mathsf{i}(\neg\ell),\ (\neg\mathbf{B}_\mathsf{i}(\ell) \wedge \neg\mathbf{B}_\mathsf{i}(\neg\ell))\}$). The case where $\mathsf{u} \not\models \varphi$ is similar and, therefore, omitted.

- Let us start by recalling that the executor agent j consider her/him-self as stubborn, given that announcing something should not affect her/his beliefs on what she/he has announced. This means that, to calculate the updated version of u', agent j applies the sub-function S as the stubborn agents do.

- Now, being $\mathsf{i} \in (\mathbf{S}_\mathsf{a} \cup \{\mathsf{j}\})$, we know from Definition 4.4 that the updated version of her/his reachable possibilities is represented by the set $\mathcal{B}^{\mathsf{u}'}_\mathsf{i} = \{\mathsf{p}' \mid \mathsf{p} \in \mathcal{B}^\mathsf{u}_\mathsf{i} \wedge \mathsf{p}' = \mathsf{S}(\mathsf{a},\mathsf{u},\ell,\mathtt{s}),\}$ (The Boolean value s is either 1, if $\mathsf{i} \in \mathbf{S}_\mathsf{a}$, or 0, when $\mathsf{i} = \mathsf{j}$).

- Following Definition 4.4 we know that each possibility in $\mathcal{B}^{\mathsf{u}'}_\mathsf{i}$ has the same fluent set of its previous version.

- Moreover, we know that an stubborn agent preserves all the edges. In fact the unfolding of the execution of S from u, when considered from an stubborn agent i's point of view, simply re-applies S to all the possibilities in $\mathcal{B}^\mathsf{u}_\mathsf{i}$. This means that if an agent reached a possibility q from another possibility p in u she/he will reach q' from p' in u'.

- From the last statement, and given that the updated version of each possibility maintains the same fluent set we can conclude that, if $\mathsf{u} \models \varphi$ then $\mathsf{u}' \models \varphi$ (similarly if $\mathsf{u} \not\models \varphi$ then $\mathsf{u}' \not\models \varphi$) with $\varphi \in$

$\{\mathbf{B}_{\mathsf{i}}(\ell),\ \mathbf{B}_{\mathsf{i}}(\neg\ell),\ (\neg\mathbf{B}_{\mathsf{i}}(\ell) \wedge \neg\mathbf{B}_{\mathsf{i}}(\neg\ell))\}$ and $\mathsf{i} \in (\mathbf{S}_{\mathsf{a}} \cup \{\mathsf{j}\})$.

(4) We identify the set of the possibilities reachable by *partial observants* agents with $\mathcal{B}^{\mathsf{u}}_{\mathbf{P}_{\mathsf{a}}}$. We also recall that this set is equal to $\mathbf{C}_{\mathbf{P}_{\mathsf{a}}}$ in u.

- Now to calculate $\mathcal{B}^{\mathsf{u}'}_{\mathbf{P}_{\mathsf{a}}}$, following Definition 4.4, we apply $\mathsf{P}(\mathsf{a},\mathsf{p})$ to every element p of $\mathcal{B}^{\mathsf{u}}_{\mathbf{P}_{\mathsf{a}}}$. This results in all the possibilities p' of $\mathcal{B}^{\mathsf{u}'}_{\mathbf{P}_{\mathsf{a}}}$ to have the same fluent set of the corresponding possibility $\mathsf{p} \in \mathcal{B}^{\mathsf{u}}_{\mathbf{P}_{\mathsf{a}}}$.

- It is easy to identify two disjoint subsets $\mathcal{B}^{\mathsf{u}'_0}_{\mathbf{P}_{\mathsf{a}}}$ and $\mathcal{B}^{\mathsf{u}'_1}_{\mathbf{P}_{\mathsf{a}}}$ of $\mathcal{B}^{\mathsf{u}'}_{\mathbf{P}_{\mathsf{a}}}$ that contains only possibility such that:
  - $\mathcal{B}^{\mathsf{u}'_0}_{\mathbf{P}_{\mathsf{a}}} \not\models \ell$;
  - $\mathcal{B}^{\mathsf{u}'_1}_{\mathbf{P}_{\mathsf{a}}} \models \ell$;
  - $(\mathcal{B}^{\mathsf{u}'_0}_{\mathbf{P}_{\mathsf{a}}} \cup \mathcal{B}^{\mathsf{u}'_1}_{\mathbf{P}_{\mathsf{a}}}) = \mathcal{B}^{\mathsf{u}'}_{\mathbf{P}_{\mathsf{a}}}$; and
  - $(\mathcal{B}^{\mathsf{u}'_0}_{\mathbf{P}_{\mathsf{a}}} \cap \mathcal{B}^{\mathsf{u}'_1}_{\mathbf{P}_{\mathsf{a}}}) = \emptyset$.

- From these two sets, following Definition 4.4 we can now construct the sets $\mathcal{B}^{\mathsf{u}'_0}_{\mathbf{P}_{\mathsf{a}},\mathsf{i}}$ and $\mathcal{B}^{\mathsf{u}'_1}_{\mathbf{P}_{\mathsf{a}},\mathsf{i}}$, with $\mathsf{i} \in (\mathbf{F}_{\mathsf{a}} \cup \{\mathsf{j}\})$, by applying the sub-functions $\chi(\mathtt{f},\mathsf{p},0)\ \forall\mathsf{p} \in \mathcal{B}^{\mathsf{u}'_0}_{\mathbf{P}_{\mathsf{a}}}$ and $\chi(\mathtt{f},\mathsf{p},1)\ \forall\mathsf{p} \in \mathcal{B}^{\mathsf{u}'_1}_{\mathbf{P}_{\mathsf{a}}}$ respectively. These two sets are simply the set of possibilities reachable from the *fully observant* agents (and the executor, considered *fully observant* by the *partially observants*) starting from $\mathcal{B}^{\mathsf{u}'_0}_{\mathbf{P}_{\mathsf{a}}}$ and $\mathcal{B}^{\mathsf{u}'_1}_{\mathbf{P}_{\mathsf{a}}}$ respectively.

- Let us note that `trustful`, `stubborn` and the executor are considered equally by the *partially observant* given that they do not identify a truth value but simply believe that the *fully observant* agents will know the truth value of the announced fluent.

- Given that the set $\mathcal{B}^{\mathsf{u}'_0}_{\mathbf{P}_{\mathsf{a}},\mathsf{i}}$ resulted from the application of the transition function from the point of view of *fully observant* agents, we know from Items 1 and 2 that for $\forall\mathsf{p} \in \mathcal{B}^{\mathsf{u}'_0}_{\mathbf{P}_{\mathsf{a}},\mathsf{i}}, \mathsf{p} \not\models \ell$.

- This implies that $\mathcal{B}^{\mathsf{u}'_0}_{\mathbf{P}_{\mathsf{a}},\mathsf{i}}$ reaches only possibilities where the interpretation of ℓ is false and, similarly, in $\mathcal{B}^{\mathsf{u}'_1}_{\mathbf{P}_{\mathsf{a}},\mathsf{i}}$ reaches only possibilities where the interpretation of ℓ is true.

- This means that $\mathcal{B}^{\mathsf{u}'_0}_{\mathbf{P}_{\mathsf{a}},\mathsf{i}} \models \neg\ell$ and $\mathcal{B}^{\mathsf{u}'_1}_{\mathbf{P}_{\mathsf{a}},\mathsf{i}} \models \ell$.

- It is easy to see, then, that $\mathcal{B}^{\mathsf{u}'_0}_{\mathbf{P}_{\mathsf{a}}} \models \mathbf{B}_{\mathsf{i}}(\neg\ell)$ being $\mathcal{B}^{\mathsf{u}'_0}_{\mathbf{P}_{\mathsf{a}},\mathsf{i}} = \{\mathsf{p} \mid \mathsf{p} \in$

$\bigcup_{\mathsf{q}\in\mathcal{B}^{\mathsf{u}'_0}_{\mathbf{P}_\mathsf{a}}} \mathsf{q}(\mathsf{i})\}$ (and similarly $\mathcal{B}^{\mathsf{u}'_1}_{\mathbf{P}_\mathsf{a}} \models \mathbf{B}_\mathsf{i}(\ell)$).

- Finally, being $\mathcal{B}^{\mathsf{u}'}_{\mathbf{P}_\mathsf{a}} = \mathcal{B}^{\mathsf{u}'_0}_{\mathbf{P}_\mathsf{a}} \cup \mathcal{B}^{\mathsf{u}'_1}_{\mathbf{P}_\mathsf{a}}$ we can conclude that $\mathcal{B}^{\mathsf{u}'}_{\mathbf{P}_\mathsf{a}} \models \mathbf{B}_\mathsf{i}(\neg\ell) \vee \mathbf{B}_\mathsf{i}(\ell)^a$ and therefore $\mathsf{u}' \models \mathbf{C}_{\mathbf{P}_\mathsf{a}}(\mathbf{B}_\mathsf{i}(\neg\ell) \vee \mathbf{B}_\mathsf{i}(\ell))$.

(5) Let us now illustrate the proof of Item **(5)**.

- First, to avoid unnecessary clutter let us use i) $\mathbf{V}_\mathsf{a}$ to indicate the set of the *observant* agents, *i.e.*, $\mathbf{V}_\mathsf{a} = (\mathbf{F}_\mathsf{a} \cup \mathbf{P}_\mathsf{a} \cup \{\mathsf{j}\})$ and; ii) i to indicate a `doubtful` agent, *i.e.*, $\mathsf{i} \in \mathbf{D}_\mathsf{a}$.

- We then identify the set of all the possibilities reached by the *observant* agents in u as $\mathcal{B}^{\mathsf{u}}_{\mathbf{V}_\mathsf{a}}$.

- Next, we re-apply the reachability function following the beliefs of the `doubtful` agents. This means that the set of beliefs of a `doubtful` agent i, from the point of view of the *observant* ones, is represented by the set $\mathcal{B}^{\mathsf{u}}_{\mathbf{V}_\mathsf{a},\mathsf{i}} = \{\mathsf{p} \mid \mathsf{p} \in \mathcal{B}^{\mathsf{q}}_\mathsf{i} \wedge \mathsf{q} \in \mathcal{B}^{\mathsf{u}}_\mathsf{i}\}$.

- Now to calculate $\mathcal{B}^{\mathsf{u}'}_{\mathbf{V}_\mathsf{a},\mathsf{i}}$, following Definition 4.4, we apply both $\chi(\mathtt{f},\mathsf{p},0)$ and $\chi(\mathtt{f},\mathsf{p},1)$ to every element p of $\mathcal{B}^{\mathsf{u}}_{\mathbf{V}_\mathsf{a},\mathsf{i}}$.

- This means that the set of updated beliefs of a `doubtful` agent, from the point of view of the *observant* ones, is represented by the union of the sets $\mathcal{B}^{\mathsf{u}'_0}_{\mathbf{V}_\mathsf{a},\mathsf{i}} = \{\mathsf{p}' \mid \mathsf{p}'(\mathcal{F}) = ((\mathsf{p}(\mathcal{F}) \setminus \{\mathtt{f}\}) \cup \{\neg\mathtt{f}\}) \wedge \mathsf{p} \in \mathcal{B}^{\mathsf{u}}_{\mathbf{V}_\mathsf{a},\mathsf{i}}\}$ and $\mathcal{B}^{\mathsf{u}'_1}_{\mathbf{V}_\mathsf{a},\mathsf{i}} = \{\mathsf{p}' \mid \mathsf{p}'(\mathcal{F}) = ((\mathsf{p}(\mathcal{F}) \setminus \{\neg\mathtt{f}\}) \cup \{\mathtt{f}\}) \wedge \mathsf{p} \in \mathcal{B}^{\mathsf{u}}_{\mathbf{V}_\mathsf{a},\mathsf{i}}\}$. It is important to notice that the truth value of the fluent `f` in the set of possibilities $\mathcal{B}^{\mathsf{u}'}_{\mathbf{V}_\mathsf{a}}$ is not important as the application of $\chi(\mathtt{f},\mathsf{p},0/1)$ on all these possibilities forces their updated version to set the truth value of $\mathtt{f} = 0/1$.

- As all the reached possibility from $\mathcal{B}^{\mathsf{u}'_0}_{\mathbf{V}_\mathsf{a},\mathsf{i}}$ and $\mathcal{B}^{\mathsf{u}'_0}_{\mathbf{V}_\mathsf{a},\mathsf{i}}$ have the truth value of `f` set to 0 and 1 respectively we can easily derive that the former entails $\neg\mathtt{f}$, while the latter entails `f`.

- Moreover, being $\mathcal{B}^{\mathsf{u}'}_{\mathbf{V}_\mathsf{a},\mathsf{i}} = (\mathcal{B}^{\mathsf{u}'_0}_{\mathbf{V}_\mathsf{a},\mathsf{i}} \cup \mathcal{B}^{\mathsf{u}'_1}_{\mathbf{V}_\mathsf{a},\mathsf{i}})$, we know that $\mathcal{B}^{\mathsf{u}'}_{\mathbf{V}_\mathsf{a},\mathsf{i}} \not\models \mathtt{f}$ and $\mathcal{B}^{\mathsf{u}'}_{\mathbf{V}_\mathsf{a},\mathsf{i}} \not\models \neg\mathtt{f}$. This is true because the subset $\mathcal{B}^{\mathsf{u}'_0}_{\mathbf{V}_\mathsf{a},\mathsf{i}} \not\models \mathtt{f}$ while $\mathcal{B}^{\mathsf{u}'_1}_{\mathbf{V}_\mathsf{a},\mathsf{i}} \not\models \neg\mathtt{f}$.

- Recalling that, as shown in Appendix A.1, the set $\mathcal{B}^{\mathsf{u}}_{\alpha,\mathsf{i}}$ corresponds to the possibilities reached by $\mathbf{C}_\alpha(\mathbf{B}_\mathsf{i})$, where $\alpha \subseteq D(\mathcal{AG})$ and $\mathsf{i} \in D(\mathcal{AG})$,

it is clear that the set $\mathcal{B}^{\mathsf{u}'}_{\mathbf{V}_\mathsf{a},\mathsf{i}}$ corresponds to the possibilities reached by $\mathbf{C}_{\mathbf{V}_\mathsf{a}}(\mathbf{B}_\mathsf{i})$ starting from u'.

- Since the set identified in the last item can derive both $\mathtt{f}$ and $\neg\mathtt{f}$, following the entailment rules of Definition 2.11, we can infer that both $\mathbf{C}_{\mathbf{V}_\mathsf{a}}(\neg\mathbf{B}_\mathsf{i}(\neg\mathtt{f}))$ and $\mathbf{C}_{\mathbf{V}_\mathsf{a}}(\neg\mathbf{B}_\mathsf{i}(\mathtt{f}))$ hold.

(6) Finally, let us prove Item **(6)**.

- When an agent $\mathsf{o} \in \mathbf{O}_\mathsf{a}$, from Definition 4.4, we know that $\mathsf{p}'(\mathsf{o}) = \mathsf{p}(\mathsf{o})$. This means that, independently from the how a possibility p' has been updated, the point of view any oblivious agent o from p' is equal to the one that the point of view of o from p.

- This implies that, $\forall \mathsf{p}' \in \mathcal{B}^{\mathsf{u}'}_\mathsf{i}$ with $\mathsf{i} \in D(\mathcal{AG})$, $\mathsf{p}'(\mathsf{o}) = \mathsf{p}(\mathsf{o})$ where $\mathsf{o} \in \mathbf{O}_\mathsf{a}$.

- This means that for every element in $\mathcal{B}^{\mathsf{u}}_\mathsf{i}$ we have an updated version in $\mathcal{B}^{\mathsf{u}\prime}_\mathsf{i}$ that has the same reachability function for each oblivious agent o.

- Then, it is easy to see, that $\mathcal{B}^{\mathsf{u}}_{\mathsf{i},\mathsf{o}} = \mathcal{B}^{\mathsf{u}'}_{\mathsf{i},\mathsf{o}}$ and, therefore, that these two sets contain the same possibilities.

- Given that the two sets of possibilities are the same, it means that the reachability functions that they represent are the same. Being the two functions the same it means that given a belief formula φ, $\mathsf{u} \models \mathbf{B}_\mathsf{i}(\mathbf{B}_\mathsf{o}(\varphi))$ *iff* $\mathsf{u}' \models \mathbf{B}_\mathsf{i}(\mathbf{B}_\mathsf{o}(\varphi))$.

- Finally, we can conclude that if $\mathsf{u} \models \mathbf{B}_\mathsf{i}(\mathbf{B}_\mathsf{o}(\varphi))$ then $\mathsf{u}' \models \mathbf{B}_\mathsf{i}(\mathbf{B}_\mathsf{o}(\varphi))$.

□

[a] The two sets are completely disjoint as one only contains possibilities that entail ℓ while the other only possibilities that do not. This means that does not exist any fully-observant-edge between possibilities that belongs in two different sets.

A.5 Proofs of Propositions 5.1 to 5.3

A.5.1 Abbreviations

To avoid unnecessary clutter instead of using the predicate `pos_w`(T, R, P) to identify a generic possibility we will write `pos`(`u`) where the lowercase letter in `typewriter font` (generally `u`, `v` or `p`) identifies a generic triple (T, R, P). Whenever possible we will present a more "concrete" version of the ASP rules by removing parts of the rule that are not necessary to capture its semantics. For example, the rule for entailing a fluent literal `f`, that in ASP has the generic form:

`entails`(T, R, P, F) :- `time`(T), `holds`(T, R, P, F), `pos_w`(T, R, P), `fluent`(F).

will be rewritten as:

$$\texttt{entails}(\texttt{u},\texttt{f}) \text{ :- } \texttt{holds}(\texttt{u},\texttt{f}),\ \texttt{fluent}(\texttt{f}).$$

Moreover, let us make use of the notations Γ and Φ to identify PLATO's and $m\mathcal{A}^\rho$'s transition function respectively. In the following proofs, we will use $\texttt{p}'$ instead of $\Gamma(\texttt{a},\texttt{p})$ or $\Phi(\texttt{a},\texttt{p})$ when this does not cause ambiguity and make use of the compact notation $\texttt{u}(\mathcal{F}) = \{\texttt{f} \mid \texttt{f} \in D(\mathcal{F}) \wedge \texttt{u} \models \texttt{f}\} \cup \{\neg\texttt{f} \mid \texttt{f} \in D(\mathcal{F}) \wedge \texttt{u} \not\models \texttt{f}\}$.

A.5.2 PLATO Entailment Correctness

As a first step we need to prove that the entailment in PLATO is correct with respect to the one introduced in Definition 2.11. To do that we will identify the rules in PLATO that correspond with an entailment rule in $m\mathcal{A}^\rho$ (from Section 5.3.1) and prove their correctness. For the sake of readability let us quickly re-introduce the entailment rules for possibilities used by $m\mathcal{A}^\rho$. Let a domain D, the belief formulae $\varphi, \varphi_1, \varphi_2 \in D(\mathcal{BF})$, a fluent literal $\texttt{f} \in D(\mathcal{F})$, an agent $\texttt{i} \in D(\mathcal{AG})$, a group of agents $\alpha \subseteq D(\mathcal{AG})$, and a possibility $\texttt{u} \in D(\mathcal{S})$ be given. The entailment in $m\mathcal{A}^\rho$ is defined as follows:

A. $\texttt{u} \models \texttt{f}$ if $\texttt{u}(\texttt{f}) = 1$;

B. $\texttt{u} \models \mathbf{B}_\texttt{i}(\varphi)$ if for each $\texttt{v} \in \texttt{u}(\texttt{i})$, $\texttt{v} \models \varphi$;

C. $\mathsf{u} \models \neg\varphi$ if $\mathsf{u} \not\models \varphi$;

D. $\mathsf{u} \models \varphi_1 \vee \varphi_2$ if $\mathsf{u} \models \varphi_1$ or $\mathsf{u} \models \varphi_2$;

E. $\mathsf{u} \models \varphi_1 \wedge \varphi_2$ if $\mathsf{u} \models \varphi_1$ and $\mathsf{u} \models \varphi_2$;

F. $\mathsf{u} \models \mathbf{E}_\alpha\varphi$ if $\mathsf{u} \models \mathbf{B}_\mathsf{i}(\varphi)$ for all $\mathsf{i} \in \alpha$;

G. $\mathsf{u} \models \mathbf{C}_\alpha\varphi$ if $\mathsf{u} \models \mathbf{E}^k_\alpha\varphi$ for every $k \geq 0$, where $\mathbf{E}^0_\alpha\varphi = \varphi$ and $\mathbf{E}^{k+1}_\alpha\varphi = \mathbf{E}_\alpha(\mathbf{E}^k_\alpha\varphi)$.

Proof of Proposition 5.1 To prove that the ASP encoding of the entailment is correct we will identify each entailment rule with a rule of PLATO.

- Rule *A* corresponds to:

  1. `entails(u, f) :- holds(u, f), fluent(f).`
  2. `entails(u, ¬f) :- holds(u, ¬f), fluent(f).`

  Let us note that the predicate `holds` correctness is derived from Propositions 5.2 and 5.3 (shown later). In fact, being the construction of the initial state and the update function correct, it is straightforward to see that $\forall \mathtt{f} \in D(\mathcal{F})$ and $\forall \mathsf{u} \in D(\mathcal{S})$ the predicate `holds(u, f)` is true *iff* $\mathsf{u}(\mathtt{f}) = 1$ while `holds(u, ¬f)` is true *iff* $\mathsf{u}(\mathtt{f}) = 0$.

- Rule *B* corresponds to:

  3. `not_entails(u, b(i, φ)) :- not entails(v, φ), believes(u, v, i).`
  4. `entails(u, b(i, φ)) :- not not_entails(u, b(i, φ)).`

  Similarly to the previous point, following Propositions 5.2 and 5.3, we can derive the correctness of the predicate `believes` and consequently the correctness of `reaches`. Moreover, for this case, we used an auxiliary predicate `not_entails` (ASP Rule *3*) that checks whether a given formula φ is not entailed by a possibility `v`. Namely we calculate the set $\mathcal{U}$ s.t. $\nexists \mathsf{u} \in \mathcal{U}, \mathsf{u} \not\models \varphi$. This can be rewritten as $\forall \mathsf{u} \in \mathcal{U}, \mathsf{u} \models \varphi$. Hence, for formulae of the type `b(i, φ)` we require that all of the possibilities believed by `i` do entail φ as in Rule *B*.

- Rules *C*, *D*, and *E* correspond to ASP Rules *5*, *6*-*7*, and *8*, respectively.

  5. `entails(u, neg(φ)) :- not entails(u, φ).`
  6. `entails(u, or(φ1, φ2)) :- entails(u, φ1).`
  7. `entails(u, or(φ1, φ2)) :- entails(u, φ2).`
  8. `entails(u, and(φ1, φ2)) :- entails(u, φ1), entails(u, φ2).`

These $m\mathcal{A}^\rho$ and ASP Rules represent the inductive steps of the entailment in $m\mathcal{A}^\rho$ and PLATO respectively, and it is straightforward to check their correspondence. The base cases are Rule *A* for $m\mathcal{A}^\rho$ and ASP Rules *1*, *2* for PLATO.

- Rule *F* is used to ease the writing of Rule *G* without adding any semantic to the entailment and was not necessary to transpose. The formula $\mathbf{E}_\alpha\varphi$ is, in fact, just a rewriting of $\bigwedge_{\mathsf{i}\in\alpha} \mathbf{B}_\mathsf{i}(\varphi)$.
- Rule *G* corresponds to ASP Rule *10*.

9. $\texttt{not\_entails}(\mathsf{u}, \mathsf{c}(\alpha, \varphi))$:- $\texttt{not entails}(\mathsf{v}, \varphi)$, $\texttt{reaches}(\mathsf{u}, \mathsf{v}, \mathsf{i})$.

10. $\texttt{entails}(\mathsf{u}, \mathsf{c}(\alpha, \varphi))$:- $\texttt{not not\_entails}(\mathsf{u}, \mathsf{c}(\alpha, \varphi))$.

Similarly to ASP Rule *4* for formulae of the type $\mathsf{c}(\alpha, \varphi)$ we require that all of the possibilities *reached* by α do entail φ. This is achieved through an auxiliary predicate `not_entails` (ASP Rule *9*) that checks whether a given formula φ is not entailed by a possibility v that is *reached* by α.

□

A.5.3 PLATO Initial State Construction Correctness

As already mentioned, the initial state description in $m\mathcal{A}^\rho$ must model a finitary **S5**-theory to ensure a finite number (up to bisimulation) of e-states. that can satisfy the initial conditions [Son et al., 2014]. For the sake of readability, let formally introduce the concept of Finitary **S5**.

Definition 1.1: Finitary S5-theory [Son et al., 2014]

Let a domain D, a fluent formula $\phi \in D$, and an agent $\mathsf{i} \in D(\mathcal{AG})$ be given. A *finitary **S5**-theory* is a collection of formulae of the form:

$$(i)\ \phi \qquad (ii)\ \mathbf{C}_{\mathcal{AG}}(\phi)$$

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

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

Proof of Proposition 5.2 To prove that the initial state generated in PLATO is equal to the one derived in $m\mathcal{A}^\rho$ we will show that PLATO has the same behavior as $m\mathcal{A}^\rho$ for each type of initial condition (formulae (i)–(iv)).

(ii) For a clearer proof let us start from the second type of condition, *i.e.*,

$\mathbf{C}_{\mathcal{AG}}(\phi)$. These formulae are used to determine the set of possible worlds that are contained in the initial e-state. A fluent literal f is *initially known* if there exists a formula $\mathbf{C}_{\mathcal{AG}}(\mathtt{f})$ or $\mathbf{C}_{\mathcal{AG}}(\neg\mathtt{f})$. In the former case, all the initial possible world must derive that f is true, whereas in the latter that f is false. If there are no such formulae for f, then it is said to be *initially unknown*.

Following Definition A.1 $m\mathcal{A}^{\rho}$ initial e-state contains all the worlds s.t.: *(i)* are consistent in their fluents' truth value; *(ii)* entail the correct truth value for each *initially known* fluent literal; and *(iii)* generate all the different combinations of the *initially unknown* fluents. In the same manner, PLATO determines the set of possible worlds (*i.e.*, pos_w) through the following rules:

11. unknown_init(ℓ) :- not init($\mathbf{C}_{\mathcal{AG}}(\ell)$), fluent($\ell$).
12. initial_dim(2**K) :- K = {fluent(ℓ): unknown_init(ℓ)}.
13. pos_w(1..K) :- initial_dim(K).
14. holds(u, ℓ) :- init($\mathbf{C}_{\mathcal{AG}}(\ell)$), pos_w(u), fluent($\ell$).
15. K/2 { holds(u, f) : pos_w(u)} K/2 :- unknown_init(f), initial_dim(K).
16. K/2 {not holds(u, f) : pos_w(u)} K/2 :- unknown_init(f), initial_dim(K).

Where ℓ can be either f or $\neg$f and the facts init($\mathbf{C}_{\mathcal{AG}}(\ell)$) are given.

(i) Formulae of type (i) are used to identify which possibility among the initial ones (determined by the previous step) identifies the pointed world. In particular, this type of condition is used to express the truth values of the fluents in the initial pointed world. That is, every formula expressed through conditions of this type must be true in the initial pointed world. In PLATO this type of condition is expressed as follows:

17. pointed(u) :- init(ℓ), pos_w(u), holds(u, ℓ), fluent(ℓ).

Where ℓ can be either f or $\neg$f and the facts init(ℓ) are given.

(iii) Formulae of the form $\mathbf{C}_{\mathcal{AG}}(\mathbf{B}_{\mathsf{i}}(\phi) \vee \mathbf{B}_{\mathsf{i}}(\neg\phi))$ are used to filter out the edges of the initial state. In particular, during the initial state construction in $m\mathcal{A}^{\rho}$ formulae of this type remove the edges, labeled with i, that link two possible worlds that "disagree" on the truth value of ϕ. This is also done in PLATO using the following rules:

18. not_b_init(u, v, i) :- pos_w({u, v}), init($\mathbf{C}$(or(b(i, ϕ), b(i, $\neg\phi$)))).
19. not_b_init(u, v, i) :- pos_w({u, v}), init($\mathbf{C}$(or(b(i, ϕ), b(i, $\neg\phi$)))).
20. believes(u, v, i) :- pos_w({u, v}), not not_b_init(u, v, i).

(*iv*) Formulae of the type (*iv*) do not filter out any other edges. Since the construction of the initial state is achieved by removing the edges of a complete graph—*i.e.*, being $\mathcal{G}$ the set of initial possibilities, $\forall \mathsf{u} \in \mathcal{G}, \forall \mathsf{i} \in \mathcal{AG}$ we have that $\mathsf{u}(\mathsf{i}) = \mathcal{G}$. We can observe that this type of formulae does not contribute to this filtering, hence we do not consider them in the initial state generation in PLATO.

Let us note that, being formulae (*i*)–(*iv*) the only ones allowed, PLATO constructs the initial state only using ASP Rules *11–20*. □

A.5.4 PLATO Transition Function Correctness

To prove the correctness of the ASP-based e-state update we will prove the correspondence between PLATO and $m\mathcal{A}^{\rho}$ for ontic and epistemic (*i.e.*, sensing and announcement) actions, separately. Before proving each action type we will briefly re-illustrate its transition function as defined in Definition 2.12. Once again, let a domain D, its set of action instances $D(\mathcal{AI})$, the set $D(\mathcal{S})$ of all the e-states reachable from $D(\varphi_{ini})$ with a finite sequence of action instances, an action instance $\mathsf{a} \in D(\mathcal{AI})$, a possibility $\mathsf{u} \in D(\mathcal{S})$, and an agent $\mathsf{i} \in D(\mathcal{AG})$ be given.

The transition function $\Phi : D(\mathcal{AI}) \times D(\mathcal{S}) \to D(\mathcal{S}) \cup \{\emptyset\}$ for an executable ontic action[1] is $\Phi(\mathsf{a}, \mathsf{u}) = \mathsf{u}'$, where:

$$e(\mathsf{a}, \mathsf{u}) = \{\ell \mid (\mathsf{a} \ \mathtt{causes} \ \ell) \in D\}; \text{ and}$$

$$\overline{e(\mathsf{a}, \mathsf{u})} = \{\neg\ell \mid \ell \in e(\mathsf{a}, \mathsf{u})\} \text{ where } \neg\neg\ell \text{ is replaced by } \ell.$$

H.
$$\mathsf{u}'(\mathtt{f}) = \begin{cases} 1 & \text{if } \mathtt{f} \in (\mathsf{u}(\mathcal{F}) \setminus \overline{e(\mathsf{a}, \mathsf{u})}) \cup e(\mathsf{a}, \mathsf{u}) \\ 0 & \text{if } \neg\mathtt{f} \in (\mathsf{u}(\mathcal{F}) \setminus \overline{e(\mathsf{a}, \mathsf{u})}) \cup e(\mathsf{a}, \mathsf{u}) \end{cases}$$

I.
$$\mathsf{u}'(\mathsf{i}) = \begin{cases} \mathsf{u}(\mathsf{i}) & \text{if } \mathsf{i} \in \mathbf{O}_{\mathsf{a}} \\ \bigcup\limits_{\mathsf{w} \in \mathsf{u}(\mathsf{i})} \Phi(\mathsf{a}, \mathsf{w}) & \text{if } \mathsf{i} \in \mathbf{F}_{\mathsf{a}} \end{cases}$$

Proof of Proposition 5.3 (for ontic actions) To prove that two possibilities generated from two different transition functions, starting from equal possibilities, entail the same formulae we need to prove that the updated possibilities have the same structural properties. To show this we will identify Rules *H* and *I* with ASP rules.

[1] If a is not executable in u, then $\Phi(\mathsf{a}, \mathsf{u}) = \emptyset$.

- Rule H corresponds to:

  21. `holds(u′, ℓ) :- causes(a, ℓ), pos_w(u), pos_w(u′), plan(T, a).`

  22. `holds(u′, ℓ) :- not causes(a, ℓ), holds(u, ℓ), pos_w(u), pos_w(u′), plan(T, a).`

  Let us start by showing that the updated possibilities u' and v', generated from $\Phi(\mathsf{a}, \mathsf{u})$ and $\Gamma(\mathsf{a}, \mathsf{v})$ respectively, are equal with respect to the fluents truth value. Let us consider the case when the action `a causes f`; in this scenario $\mathsf{u}'(\mathtt{f})$ is equal to 1 (Rule H) and `holds(v′, f)` is valid (ASP Rule *21*) meaning that both u' and v' consider `f` to be true.

  Similarly, when the action `a causes ¬f` we will have that $\mathsf{u}'(\mathtt{f})$ is equal to 0 (Rule H) while the predicate `holds(v′, ¬f)` is true (ASP Rule *21*) causing `f` to be false in u' and v'.

  Finally, we need to show that the fluents that are not modified by the action have the same truth value both in u' and v'. This is easily derived in $m\mathcal{A}^{\rho}$ as in Rule H the fluents modified are only the ones that belong to the set $e(\mathsf{a}, \mathsf{u})$—namely the effects of a—while the others are preserved from $\mathsf{u}(\mathcal{F})$. On the other hand, in PLATO, this is accomplished with ASP Rule *22* that explicitly sets every fluent literal that is not an effect of a as it was in v. Given that we assumed u and v to entail the same formulae, and therefore to have the same truth value for fluents, we can conclude that also the fluents not directly modified by a have the same value in u' and v'.

- After the fluents truth value we need to prove that the beliefs update is the same in both $m\mathcal{A}^{\rho}$ and PLATO.

  – Let us start with the beliefs related to the oblivious agents. The first case of Rule I (Rule I_1) corresponds to:

    23. `believes(u′, v, i) :- believes(u, v, i), oblivious(i, a), pos_w({u, u′, v}).`

    As described in Rule I_1 an oblivious agent i, from u', believes the same set of possibilities $\mathcal{U}_{\mathsf{i}}$ that she believed in u. In PLATO the behavior of an oblivious agent i is described by ASP Rule *23* that creates a predicate `believes` from v' to each possibility that belongs to the set $\mathcal{V}_{\mathsf{i}}$ of possibilities believed by i in v. Given that, by definition, u and v must entail the same formulae we have that the sets of possibilities believed by an agent starting from u and v must be equals. In particular, this means that the sets $\mathcal{U}_{\mathsf{i}}$ and $\mathcal{V}_{\mathsf{i}}$ are the same set and, therefore, an oblivious agent's beliefs are the same starting from u' or v'.

  – Next, we will prove that the beliefs of fully observant agents are equals in u' and v'. The second case of Rule I (Rule I_2) corresponds to ASP Rule *24*.

24. `believes`(u', v', i) :- `believes`(u, v, i), `fully_obs`(i, a), `pos_w`($\{\mathsf{u}, \mathsf{u}', \mathsf{v}, \mathsf{v}'\}$).

This scenario for $m\mathcal{A}^{\rho}$ is described in Rule I_2 where it is shown how a fully observant agent i, starting from u', believes the updated version of the possibilities that she believed starting from u. The same holds for PLATO where ASP Rule *24* creates a predicate `believes` from v' to every updated version of the possibility believed by i in v. This means that a fully observant agent, that necessarily believes the same set $\mathcal{P}_{\mathsf{i}}$ of possibilities starting from u and v, believes the updated version of $\mathcal{P}_{\mathsf{i}}$ starting from u' and v'. As shown in the other points the result of both the transition functions on a possibility p is the same possibility p' and therefore the updated version of $\mathcal{P}_{\mathsf{i}}$ is equal in both $m\mathcal{A}^{\rho}$ and PLATO.

□

In what follows we will provide the proof for the transition function of an announcement action. While this update is different from the one used for sensing actions, their behavior is very similar. The only difference is that sensing actions only consider fluent literals as effects while announcements allow for entire fluent formulae. Being each fluent literal a fluent formula itself, we have that the proof for sensing actions falls under the one for announcements. That is why, for the sake of readability, we will only show that the update is correct for announcements. The transition function $\Phi : D(\mathcal{AI}) \times D(\mathcal{S}) \rightarrow D(\mathcal{S}) \cup \{\emptyset\}$ for an executable announcement action[2] is $\Phi(\mathsf{a}, \mathsf{u}) = \mathsf{u}'$, where:

$$e(\mathsf{a}, \mathsf{u}) = \begin{cases} 0 & \text{if } \mathsf{u} \models \phi \\ 1 & \text{if } \mathsf{u} \models \neg\phi \end{cases}$$

J. $$\mathsf{u}'(\mathcal{F}) = \mathsf{u}(\mathcal{F})$$

K. $$\mathsf{u}'(\mathsf{i}) = \begin{cases} \mathsf{u}(\mathsf{i}) & \text{if } \mathsf{i} \in \mathbf{O}_{\mathsf{a}} \\ \bigcup\limits_{\mathsf{w} \in \mathsf{u}(\mathsf{i})} \Phi(\mathsf{a}, \mathsf{w}) & \text{if } \mathsf{i} \in \mathbf{P}_{\mathsf{a}} \\ \bigcup\limits_{\mathsf{w} \in \mathsf{u}(\mathsf{i}):\ e(\mathsf{a},\mathsf{w}) = e(\mathsf{a},\mathsf{u})} \Phi(\mathsf{a}, \mathsf{w}) & \text{if } \mathsf{i} \in \mathbf{F}_{\mathsf{a}} \end{cases}$$

[2] If a is not executable in u, then $\Phi(\mathsf{a}, \mathsf{u}) = \emptyset$.

Proof of Proposition 5.3 (for announcement actions) As in the previous proof, we will identify Rules J and K with ASP rules.

- Rule J corresponds to:

  25. `holds`(u′, ℓ) :- `plan`(T, a), `pos_w`(u), `pos_w`(u′), `holds`(u, ℓ).

  Let us start by showing that the updated possibilities u′ and v′, generated from $\Phi(\mathsf{a}, \mathsf{u})$ and $\Gamma(\mathsf{a}, \mathsf{v})$ respectively, are equal with respect to the fluents truth value. This is easily derived: in fact in $m\mathcal{A}^{\rho}$ (Equation J) the fluents interpretation in u′ is equal to the fluents interpretation of u and in PLATO the predicates `holds` are valid on the same fluents interpretation in both v and v′ (ASP Rule *25*). Given that we assumed u and v to entail the same formulae, and therefore to have the same truth value for fluents, we can conclude that also the fluents have the same value in u′ and v′.

- After the fluents truth value we need to prove that the belief update is the same in both $m\mathcal{A}^{\rho}$ and PLATO.

  – Let us start with the beliefs related to the oblivious agents. The first case of Rule K (Rule K_1) corresponds to ASP Rule *26*.

    26. `believes`(u′, v, i) :- `believes`(u, v, i), `oblivious`(i, a), `pos_w`({u, u′, v}).

    As for the ontic actions an oblivious agent i, from u′, believes the same set of possibilities $\mathcal{U}_{\mathsf{i}}$ that she believed in u (Rule K_1) and in PLATO i `believes`, from v′, the set $\mathcal{V}_{\mathsf{i}}$ of possibilities believed by i in v (ASP Rule *26*). Given that, by definition, u and v must entail the same formulae we have that the sets of possibilities believed by an agent starting from u and v must be equals. In particular, this means that the sets $\mathcal{U}_{\mathsf{i}}$ and $\mathcal{V}_{\mathsf{i}}$ are the same set and, therefore, an oblivious agent's beliefs are the same starting from u′ or v′.

  – Next we need to show that the partially observant agents' beliefs are equals in u′ and v′. The second case of Rule K (Rule K_2) corresponds to:

    27. `believes`(u′, v′, i) :- `believes`(u, v, i), `partial_obs`(i, a), `pos_w`({u, u′, v, v′}).

    This scenario for $m\mathcal{A}^{\rho}$ is described by Rule K_2 where it is shown how a partially observant agent i, starting from u′, believes the updated version of the possibilities that she believed starting from u. The same holds for PLATO where ASP Rule *27* creates a predicate `believes` from v′ to every updated version of the possibility belived by i in v. This means that a partially observant agent, that necessarily believes the same set $\mathcal{P}_{\mathsf{i}}$ of possibilities starting from u and v, believes the updated version of $\mathcal{P}_{\mathsf{i}}$ starting from u′ and v′. As shown in the other points the result of both

the transition functions on a possibility p is the same possibility p' and therefore the updated version of $\mathcal{P}_\mathsf{i}$ is equal in both $m\mathcal{A}^\rho$ and PLATO.

– Finally, we need to prove that also the beliefs of the fully observant agents are equals in u' and v'. The third case of Rule K (Rule K_3) corresponds to:

28. `pos_w(`u'`) :- plan(T, a), pos_w(u), reach_fully(`$\mathsf{pointed}_\mathsf{u}$`, u), entails(u, `$\phi$`), entails(`$\mathsf{pointed}_\mathsf{u}$`, `$\phi$`).`

29. `pos_w(`u'`) :- plan(T, a), pos_w(u), believes(`$\mathsf{pointed}_\mathsf{u}$`, u, i), partial_obs(i, a).`

30. `pos_w(`u'`) :- plan(T, a), pos_w({u,v}), believes(`$\mathsf{pointed}_\mathsf{u}$`, v, i), partial_obs(i, a), reach_not_oblivious(v, u).`

31. `believes(`u'`, `v'`, i) :- believes(u, v, i), fully_obs(i, a), holds({u, v}, `ℓ`),pos_w({u, `u'`, v, `v'`}).`

Given that $\Phi(\mathsf{a}, \mathsf{u})$ is assumed to be applied starting from the pointed world we have that a fully observant agent, starting from the pointed possibility, only believes possibilities where ϕ has the same truth value that has in the pointed one. This case is matched exactly in PLATO by the combination of ASP Rules *28* and *31*. On the other hand, if a world is reached by a fully observant agent not directly from the pointed world—*i.e.*, it is reached by a fully observant through a path of partially and fully observant agents that starts with a partially observant one—its updated version will only have fully observant edges to the updated possibilities with the same interpretation of ϕ. This is because Rule K_2 is firstly applied and finally (possibly after other applications of Rule K) Rule K_3 is used. In fact, by applying Rule K_2 first, Φ is recursively applied on both possibilities that have and do not have the same interpretation of ϕ with respect to the pointed world. It is straightforward to see that this rule is transposed in PLATO through the combination of ASP Rules *30* and *31*.

□

No one knows everything,
but true wisdom is to know whom to ask.

— Students' proverb
in Moss, *Dynamic Epistemic Logic*
[Moss, 2015]

Bibliography

Peter Aczel. Non-well-founded sets. CSLI Lecture Notes, 14, 1988.

Martin Allen and Shlomo Zilberstein. Complexity of decentralized control: Special cases. In *23rd Annual Conference on Neural Information Processing Systems 2009, 7-10 December , Vancouver, British Columbia, Canada*, pages 19–27. Curran Associates, Inc., 2009. URL `https://proceedings.neurips.cc/paper/2009/hash/fec8d47d412bcbeece3d9128ae855a7a-Abstract.html`.

Ken Arnold, James Gosling, and David Holmes. *The Java programming language*. Addison Wesley Professional, 2005.

Robert Aumann, Adam Brandenburger, et al. Epistemic conditions for Nash equilibrium. *ECONOMETRICA-EVANSTON ILL-*, 63:1161–1161, 1995.

Roberta Ballarin. Modern Origins of Modal Logic. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, Fall 2021 edition, 2021.

Musard Balliu, Mads Dam, and Gurvan Le Guernic. Epistemic temporal logic for information flow security. In *Proceedings of the ACM SIGPLAN 6th Workshop on Programming Languages and Analysis for Security*, PLAS '11, pages 6:1–6:12, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0830-4. doi: 10.1145/2166956.2166962.

Alexandru Baltag and Lawrence S. Moss. Logics for epistemic programs. *Synthese*, 139(2):165–224, 2004.

Alexandru Baltag and Sonja Smets. *A Qualitative Theory of Dynamic Interactive Belief Revision*, pages 813–858. Springer International Publishing, Cham, 2016. ISBN 978-3-319-20451-2. doi: 10.1007/978-3-319-20451-2_39.

Chitta Baral, Gregory Gelfond, Tran Cao Son, and Enrico Pontelli. Using answer set programming to model multi-agent scenarios involving agents' knowledge about other's knowledge. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: Volume 1 - Volume 1*, AAMAS '10, page 259–266, Richland, SC, 2010. International Foundation for Autonomous Agents and Multiagent Systems. ISBN 9780982657119.

Chitta Baral, Gregory Gelfond, Enrico Pontelli, and Tran Cao Son. An action language for multi-agent domains: Foundations. *CoRR*, abs/1511.01960, 2015. URL `http://arxiv.org/abs/1511.01960`.

Chitta Baral, Gregory Gelfond, Enrico Pontelli, and Tran Cao Son. An action language for multi-agent domains. *Artificial Intelligence*, 302:103601, 2022. ISSN 0004-3702. doi: https://doi.org/10.1016/j.artint.2021.103601. URL `https://www.sciencedirect.com/science/article/pii/S0004370221001521`.

Yoshua Bengio, Yann Lecun, and Geoffrey Hinton. Deep learning for ai. *Commun. ACM*, 64(7):58–65, June 2021. ISSN 0001-0782. doi: 10.1145/3448250.

Daniel S. Bernstein, Robert Givan, Neil Immerman, and Shlomo Zilberstein. The complexity of decentralized control of markov decision processes. *Mathematics of operations research*, 27(4):819–840, 2002.

Tarek R. Besold, Artur d'Avila Garcez, Sebastian Bader, Howard Bowman, Pedro Domingos, Pascal Hitzler, Kai-Uwe Kuehnberger, Luis C. Lamb, Daniel Lowd, Priscila Machado Vieira Lima, Leo de Penning, Gadi Pinkas, Hoifung Poon, and Gerson Zaverucha. Neural-symbolic learning and reasoning: A survey and interpretation, 2017.

Ivan Boh. *Epistemic Logic in the Later Middle Ages (1st ed.)*. Routledge, 1993. ISBN 9780203976685. doi: 10.4324/9780203976685.

T. Bolander, M.H. Jensen, and F. Schwarzentruber. Complexity results in epistemic planning. In *IJCAI International Joint Conference on Artificial Intelligence*, volume 2015-January, pages 2791–2797, 2015.

Thomas Bolander and Mikkel Birkegaard Andersen. Epistemic planning for single- and multi-agent systems. *Journal of Applied Non-Classical Logics*, 21(1):9–34, 2011. doi: 10.1016/0010-0277(83)90004-5.

Grady Booch, Francesco Fabiano, Lior Horesh, Kiran Kate, Jonathan Lenchner, Nick Linck, Andrea Loreggia, Keerthiram Murugesan, Nicholas Mattei, Francesca Rossi, and Biplav Srivastava. Thinking fast and slow in AI. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, Virtual Event*, pages 15042–15046. AAAI Press, 2021. URL `https://ojs.aaai.org/index.php/AAAI/article/view/17765`.

Michael H. Bowling, Rune M. Jensen, and Manuela M. Veloso. Multiagent planning in the presence of multiple goals. *Planning in Intelligent Systems: Aspects, Motivations and Methods, John Wiley and Sons, Inc*, 2005.

Candida Bowtell and Peter Keevash. The n-queens problem, 2021.

Alessandro Burigana, Francesco Fabiano, Agostino Dovier, and Enrico Pontelli. Modelling multi-agent epistemic planning in asp. *Theory and Practice of Logic Programming*, 20(5):593–608, 2020. doi: 10.1017/S1471068420000289.

Christer Bäckström. Expressive equivalence of planning formalisms. *Artificial Intelligence*, 76(1):17–34, 1995. ISSN 0004-3702. doi: https://doi.org/10.1016/0004-3702(94)00081-B. URL `https://www.sciencedirect.com/science/article/pii/000437029400081B`. Planning and Scheduling.

Cambridge Dictionary. plan. In *Cambridge Dictionary*. Cambridge University Press, online edition, 2021. URL `https://dictionary.cambridge.org/dictionary/english/plan`.

Jaime G. Carbonell Jr. Politics: Automated ideological reasoning. *Cognitive Science*, 2(1):27–51, 1978. doi: 10.1207/s15516709cog0201_3.

C. Castelfranchi and R. Falcone. Principles of trust for mas: cognitive anatomy, social importance, and quantification. In *Proceedings International Conference on Multi Agent Systems (Cat. No.98EX160)*, pages 72–79, 1998. doi: 10.1109/ICMAS.1998.699034.

Alexander V. Chagrov and Michael Zakharyaschev. *Modal Logic*, volume 35 of *Oxford logic guides*. Oxford University Press, 1997. ISBN 978-0-19-853779-3.

L. Chu, Seyoun Park, S. Kawamoto, Yan Wang, Yuyin Zhou, Wei Shen, Zhuotun Zhu, Yingda Xia, Lingxi Xie, Fengze Liu, Qihang Yu, D. Fouladi, S. Shayesteh, E. Zinreich, J. Graves, K. Horton, A. Yuille, R. Hruban, K. Kinzler, B. Vogelstein, and E. Fishman. Application of deep learning to pancreatic cancer detection: Lessons learned from our initial experience. *Journal of the American College of Radiology : JACR*, 16 9 Pt B:1338–1342, 2019.

Michael T. Cox. Metacognition in computation: A selected research review. *Artificial Intelligence*, 169(2):104–141, 2005. ISSN 0004-3702. doi: https://doi.org/10.1016/j.artint.2005.10.009. URL `https://www.sciencedirect.com/science/article/pii/S0004370205001530`. Special Review Issue.

Michael T. Cox and Anita Raja. *Metareasoning: Thinking about Thinking*. The MIT Press, 2011. ISBN 0262014807.

Mathijs De Weerdt and Brad Clement. Introduction to planning in multiagent systems. *Multiagent and Grid Systems*, 5(4):345–355, 2009. doi: 10.3233/MGS-2009-0133.

Mathijs De Weerdt, André Bos, Hans Tonino, and Cees Witteveen. A resource logic for multi-agent plan merging. *Annals of Mathematics and Artificial Intelligence*, 37(1-2):93–130, 2003. doi: 10.1023/A:1020236119243.

Agostino Dovier. Logic programming and bisimulation. In *ICLP*, volume 1433 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2015. URL `http://ceur-ws.org/Vol-1433`.

Agostino Dovier, Carla Piazza, and Alberto Policriti. An efficient algorithm for computing bisimulation equivalence. *Theoretical Computer Science*, 311(1-3): 221–256, 2004.

Edmund H. Durfee. *Distributed Problem Solving and Planning*, pages 118–149. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001. ISBN 978-3-540-47745-7. doi: 10.1007/3-540-47745-4_6.

Francesco Fabiano. Design of a solver for multi-agent epistemic planning. In *Proceedings 35th International Conference on Logic Programming (Technical Communications)*, pages 403–412, 2019. doi: 10.4204/EPTCS.306.54.

Francesco Fabiano. Towards a complete characterization of epistemic reasoning: the notion of trust. In *Proceedings of the 35th Italian Conference on Computational Logic*, volume 2710 of *CEUR Workshop Proceedings*, pages 21–35, Calabria, Italy (Online), October 13-15 2020. CEUR-WS.org. URL `http://ceur-ws.org/Vol-2710/paper2.pdf`.

Francesco Fabiano and Alessandro Dal Palù. An ASP approach for arteries classification in CT scans. *Journal of Logic and Computation*, 32(2):331–346, 01 2022. ISSN 0955-792X. doi: 10.1093/logcom/exab087.

Francesco Fabiano, Idriss Riouak, Agostino Dovier, and Enrico Pontelli. Non-well-founded set based multi-agent epistemic action language. In *Proceedings of the 34th Italian Conference on Computational Logic, Trieste, Italy, June 19-21, 2019*, volume 2396 of *CEUR Workshop Proceedings*, pages 242–259. CEUR-WS.org, 2019. URL `http://ceur-ws.org/Vol-2396/paper38.pdf`.

Francesco Fabiano, Alessandro Burigana, Agostino Dovier, and Enrico Pontelli. EFP 2.0: A multi-agent epistemic solver with multiple e-state representations. In *Proceedings of the Thirtieth International Conference on Automated Planning and Scheduling, Nancy, France, October 26-30, 2020*, pages 101–109. AAAI Press, 2020. URL https://aaai.org/ojs/index.php/ICAPS/article/view/6650.

Francesco Fabiano, Alessandro Burigana, Agostino Dovier, Enrico Pontelli, and Tran Cao Son. Multi-agent epistemic planning with inconsistent beliefs, trust and lies. In Duc Nghia Pham, Thanaruk Theeramunkong, Guido Governatori, and Fenrong Liu, editors, *PRICAI 2021: Trends in Artificial Intelligence - 18th Pacific Rim International Conference on Artificial Intelligence, PRICAI 2021, Hanoi, Vietnam, November 8-12, 2021, Proceedings, Part I*, volume 13031 of *Lecture Notes in Computer Science*, pages 586–597. Springer, 2021a. doi: 10.1007/978-3-030-89188-6_44.

Francesco Fabiano, Biplav Srivastava, Jonathan Lenchner, Lior Horesh, Francesca Rossi, and Marianna Bergamaschi Ganapini. E-pddl: A standardized way of defining epistemic planning problems. In *Knowledge Engineering for Planning and Scheduling*, page in press, Online, August 2021b. URL https://icaps21.icaps-conference.org/workshops/KEPS/Papers/KEPS_2021_paper_3.pdf.

Ronald Fagin, Yoram Moses, Joseph Y. Halpern, and Moshe Y. Vardi. *Reasoning About Knowledge*. MIT press, 1995. ISBN 9780262061629.

Dirk Fahland, Daniel Lübke, Jan Mendling, Hajo Reijers, Barbara Weber, Matthias Weidlich, and Stefan Zugal. Declarative versus imperative process modeling languages: The issue of understandability. In Terry Halpin, John Krogstie, Selmin Nurcan, Erik Proper, Rainer Schmidt, Pnina Soffer, and Roland Ukor, editors, *Enterprise, Business-Process and Information Systems Modeling*, pages 353–366, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg. ISBN 978-3-642-01862-6.

Richard E Fikes and Nils J. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial intelligence*, 2(3-4):189–208, 1971. doi: 10.1016/0004-3702(71)90010-5.

John H Flavell. Metacognition and cognitive monitoring: A new area of cognitive–developmental inquiry. *American psychologist*, 34(10):906, 1979. doi: 10.1037/0003-066X.34.10.906.

Nicoletta Fornara. *Interaction and communication among autonomous agents in multiagent systems*. PhD thesis, Università della Svizzera italiana, 2003.

M. Fox and D. Long. Pddl2.1: An extension to pddl for expressing temporal planning domains. *Journal of Artificial Intelligence Research*, 20:61–124, Dec 2003. ISSN 1076-9757. doi: 10.1613/jair.1129. URL http://dx.doi.org/10.1613/jair.1129.

Benjamin Franklin. Poor Richard's almanac, 1750.

Marianna Bergamaschi Ganapini, Murray Campbell, Francesco Fabiano, Lior Horesh, Jon Lenchner, Andrea Loreggia, Nicholas Mattei, Francesca Rossi, Biplav Srivastava, and Kristen Brent Venable. Thinking fast and slow in AI: the role of metacognition. *CoRR*, abs/2110.01834, 2021. URL https://arxiv.org/abs/2110.01834.

Marianna Bergamaschi Ganapini, Murray Campbell, Francesco Fabiano, Lior Horesh, Jon Lenchner, Andrea Loreggia, Nicholas Mattei, Taher Rahgooy, Francesca Rossi,

Biplav Srivastava, and Kristen Brent Venable. Combining fast and slow thinking for human-like and efficient navigation in constrained environments. *CoRR*, abs/2201.07050, 2022. URL https://arxiv.org/abs/2201.07050.

Peter Gärdenfors and David Makinson. Revisions of knowledge systems using epistemic entrenchment. In Moshe Y. Vardi, editor, *Proceedings of the 2nd Conference on Theoretical Aspects of Reasoning about Knowledge, Pacific Grove, CA, USA, March 1988*, pages 83–95. Morgan Kaufmann, 1988.

Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. Multi-shot asp solving with clingo. *Theory and Practice of Logic Programming*, 19(1): 27–82, 2019.

Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In Robert Kowalski, Bowen, and Kenneth, editors, *Proceedings of International Logic Programming Conference and Symposium*, pages 1070–1080. MIT Press, 1988. URL http://www.cs.utexas.edu/users/ai-lab?gel88.

Michael Gelfond and Vladimir Lifschitz. Action languages. *Electron. Trans. Artif. Intell.*, 2:193–210, 1998. URL http://www.ep.liu.se/ej/etai/1998/007/.

J. Gerbrandy and W. Groeneveld. Reasoning about information change. *Journal of Logic, Language and Information*, 6(2):147–169, 1997. doi: 10.1023/A:1008222603071.

Jelle Gerbrandy. *Bisimulations on planet Kripke*. Inst. for Logic, Language and Computation, Univ. van Amsterdam, 1999.

Lakemeyer Gerhard and Levesque Hector J. Only knowing. In Hans van Ditmarsch, Wiebe van der Hoek, Joseph Y. Halpern, and Barteld Kooi, editors, *Handbook of Epistemic Logic*, chapter 2, pages 55–76. College Publications, 2015. ISBN 978-1-84890-158-2.

Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning: theory and practice*. Elsevier, 2004.

G. Gigerenzer and H. Brighton. Homo heuristicus: why biased minds make better inferences. *Top Cogn Sci*, 1(1):107–143, Jan 2009. doi: 10.1111/j.1756-8765.2008.01006.x.

Claudia V. Goldman and Shlomo Zilberstein. Decentralized control of cooperative systems: Categorization and complexity analysis. *J. Artif. Intell. Res.(JAIR)*, 22:143–174, 2004. doi: 10.1613/jair.1427.

Michael S. A. Graziano. *Consciousness and the Social Brain*. Oxford University Press, 2013.

Michael S. A. Graziano, Arvid Guterstam, Branden J. Bio, and Andrew I. Wilterson. Toward a standard model of consciousness: Reconciling the attention schema, global workspace, higher-order thought, and illusionist theories. *Cognitive Neuropsychology*, 37(3-4):155–172, 2020. doi: 10.1080/02643294.2019.1670630.

Carlos Guestrin, Daphne Koller, and Ronald Parr. Multiagent planning with factored MDPs. In *Advances in Neural Information Processing Systems 14, December 3-8, 2001, Vancouver, British Columbia, Canada*, pages 1523–1530. MIT Press, 2001. URL https://proceedings.neurips.cc/paper/2001/hash/7af6266cc52234b5aa339b16695f7fc4-Abstract.html.

Bob Hanson. Sudoku Assistant/Solver. https://www.stolaf.edu/people/hansonr/sudoku/, September 2021.

Yuval N. Harari. *Sapiens: a brief history of humankind*. Harper, 2015, 2015. URL https://search.library.wisc.edu/catalog/9910419687402121.

Stephen Hawking and Leonard Mlodinow. *The Grand Design*. Bantam Books, 2010. ISBN 0-553-80537-1.

Malte Helmert. Concise finite-domain representations for pddl planning tasks. *Artif. Intell.*, 173(5–6):503–535, April 2009. ISSN 0004-3702. doi: 10.1016/j.artint.2008.10.013.

Andreas Herzig, Jérôme Lang, and Pierre Marquis. Action progression and revision in multiagent belief structures. In *Sixth Workshop on Nonmonotonic Reasoning, Action, and Change (NRAC 2005)*. Citeseer, 2005.

Andreas Herzig, Emiliano Lorini, Jomi Fred Hübner, and Laurent Vercouter. A logic of trust and reputation. *Log. J. IGPL*, 18(1):214–244, 2010. doi: 10.1093/jigpal/jzp077.

Jaakko Hintikka. *Knowledge and Belief: An Introduction to the Logic of the Two Notions*. Ithaca: Cornell University Press, 1962.

Xiao Huang, Biqing Fang, Hai Wan, and Yongmei Liu. A general multi-agent epistemic planner based on higher-order belief change. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pages 1093–1101. ijcai.org, 2017. doi: 10.24963/ijcai.2017/152.

Martin Holm Jensen. *Epistemic and doxastic planning*. Technical University of Denmark, Applied Mathematics and Computer Science, 2014.

Daniel Kahneman. *Thinking, Fast and Slow*. Farrar, Straus and Giroux, 2011. ISBN 978-0374275631.

Vaibhav Katewa. *Analysis and design of multi-agent systems under communication and privacy constraints*. University of Notre Dame, 2017.

Pascal Kerschke, Holger H. Hoos, Frank Neumann, and Heike Trautmann. Automated algorithm selection: Survey and perspectives. *Evol. Comput.*, 27(1):3–45, 2019. doi: 10.1162/evco_a_00242.

Emil Keyder and Héctor Geffner. Heuristics for planning with action costs revisited. In *Proceedings of the 2008 Conference on ECAI 2008: 18th European Conference on Artificial Intelligence*, pages 588–592, Amsterdam, The Netherlands, The Netherlands, 2008. IOS Press. ISBN 978-1-58603-891-5.

Filippos Kominis and Hector Geffner. Beliefs in multiagent planning: From one agent to many. In *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling, ICAPS 2015, Jerusalem, Israel, June 7-11, 2015*, pages 147–155. AAAI Press, 2015. URL http://www.aaai.org/ocs/index.php/ICAPS/ICAPS15/paper/view/10617.

Iuliia Kotseruba and John K. Tsotsos. 40 years of cognitive architectures: core cognitive abilities and practical applications. *Artificial Intelligence Review*, 53(1): 17–94, Jan 2020. doi: 10.1007/s10462-018-9646-y.

Jerald D. Kralik, Jee Hang Lee, Paul S. Rosenbloom, Philip C. Jackson, Susan L. Epstein, Oscar J. Romero, Ricardo Sanz, Othalia Larue, Hedda R. Schmidtke, Sang Wan Lee, and Keith McGreggor. Metacognition for a common model of cognition. *Procedia Computer Science*, 145:730–739, 2018. ISSN 1877-0509. doi: https://doi.org/10.1016/j.procs.2018.11.046. URL `https://www.sciencedirect.com/science/article/pii/S1877050918323329`. Postproceedings of the 9th Annual International Conference on Biologically Inspired Cognitive Architectures, BICA 2018 (Ninth Annual Meeting of the BICA Society), held August 22-24, 2018 in Prague, Czech Republic.

Saul A. Kripke. Semantical analysis of modal logic i normal modal propositional calculi. *Mathematical Logic Quarterly*, 9(5-6):67–96, 1963. doi: 10.1002/malq.19630090502.

Ugur Kuter, Dana S. Nau, Elnatan Reisner, and Robert P. Goldman. Using classical planners to solve nondeterministic planning problems. In *Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling, ICAPS 2008, Sydney, Australia, September 14-18, 2008*, pages 190–197. AAAI, 2008. URL `http://www.aaai.org/Library/ICAPS/2008/icaps08-024.php`.

Heiner Lasi, Peter Fettke, Hans-Georg Kemper, Thomas Feld, and Michael Hoffmann. Industry 4.0. *Business & information systems engineering*, 6(4):239–242, 2014. doi: 10.1007/s12599-014-0334-4.

Tiep Le, Francesco Fabiano, Tran Cao Son, and Enrico Pontelli. EFP and PG-EFP: Epistemic forward search planners in multi-agent domains. In *Proceedings of the Twenty-Eighth International Conference on Automated Planning and Scheduling*, pages 161–170, Delft, The Netherlands, June 24–29 2018. AAAI Press. ISBN 978-1-57735-797-1. URL `https://aaai.org/ocs/index.php/ICAPS/ICAPS18/paper/view/17733`.

Vladimir Lifschitz. What is answer set programming? In *Proceedings of the 23rd National Conference on Artificial Intelligence - Volume 3*, AAAI'08, page 1594–1597. AAAI Press, 2008. ISBN 9781577353683.

Nir Lipovetzky and Hector Geffner. Width-based algorithms for classical planning: New results. In Torsten Schaub, Gerhard Friedrich, and Barry O'Sullivan, editors, *ECAI 2014 - 21st European Conference on Artificial Intelligence, 18-22 August 2014, Prague, Czech Republic - Including Prestigious Applications of Intelligent Systems (PAIS 2014)*, volume 263 of *Frontiers in Artificial Intelligence and Applications*, pages 1059–1060. IOS Press, 2014. doi: 10.3233/978-1-61499-419-0-1059.

Nir Lipovetzky and Hector Geffner. Best-first width search: Exploration and exploitation in classical planning. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, pages 3590–3596, San Francisco, California, USA, February 4-9 2017. URL `http://aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14862`.

Michael L. Littman. Markov games as a framework for multi-agent reinforcement learning. In *Machine Learning Proceedings 1994*, pages 157–163. Elsevier, 1994. doi: 10.1016/B978-1-55860-335-6.50027-1.

Gary Marcus. The next decade in ai: Four steps towards robust artificial intelligence, 2020.

John McCarthy, Marvin L. Minsky, Nathaniel Rochester, and Claude E. Shannon. A proposal for the dartmouth summer research project on artificial intelligence, august 31, 1955. *AI Magazine*, 27(4):12, Dec. 2006. doi: 10.1609/aimag.v27i4.1904. URL https://ojs.aaai.org/index.php/aimagazine/article/view/1904.

Drew McDermott, Malik Ghallab, Craig Knoblock, David Wilkins, Anthony Barrett, Dave Christianson, Marc Friedman, Chung Kwok, Keith Golden, Scott Penberthy, David Smith, Ying Sun, and Daniel Weld. PDDL - the planning domain definition language. Technical report, Technical Report, 1998.

Fiona McNeill and Alan Bundy. Facilitating interaction between virtual agents by changing ontological representation. In *Encyclopedia of E-Business Development and Management in the Global Economy*, pages 934–941. IGI Global, 2010.

John-Jules Ch. Meyer. *Modal Epistemic and Doxastic Logic*, pages 1–38. Springer Netherlands, Dordrecht, 2003. ISBN 978-94-017-4524-6. doi: 10.1007/978-94-017-4524-6_1.

Lawrence S. Moss. Dynamic epistemic logic. In Hans van Ditmarsch, Wiebe van der Hoek, Joseph Y. Halpern, and Barteld Kooi, editors, *Handbook of Epistemic Logic*, chapter 6, pages 262–312. College Publications, 2015. ISBN 978-1-84890-158-2.

Andrzej Mostowski. An undecidable arithmetical statement. *Fundamenta Mathematicae*, 36(1):143–164, 1949. doi: 10.4064/fm-36-1-143-164. URL http://eudml.org/doc/213187.

Christian J. Muise, Vaishak Belle, Paolo Felli, Sheila A. McIlraith, Tim Miller, Adrian R. Pearce, and Liz Sonenberg. Planning over multi-agent epistemic states: A classical planning approach. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA*, pages 3327–3334. AAAI Press, 2015. URL http://www.aaai.org/ocs/index.php/AAAI/AAAI15/paper/view/9974.

Thomas O. Nelson. Metamemory: A theoretical framework and new findings. *Psychology of Learning and Motivation*, 26:125–173, 1990. ISSN 0079-7421. doi: https://doi.org/10.1016/S0079-7421(08)60053-5. URL https://www.sciencedirect.com/science/article/pii/S0079742108600535.

Nicholas Nethercote and Julian Seward. Valgrind: A framework for heavyweight dynamic binary instrumentation. *SIGPLAN Not.*, 42(6):89–100, jun 2007. ISSN 0362-1340. doi: 10.1145/1273442.1250746.

Robert Paige and Robert E Tarjan. Three partition refinement algorithms. *SIAM Journal on Computing*, 16(6):973–989, 1987.

Edwin P. D. Pednault. ADL and the State-Transition Model of Action. *Journal of Logic and Computation*, 4(5):467–512, 10 1994. ISSN 0955-792X. doi: 10.1093/logcom/4.5.467.

Ingmar Posner. Robots thinking fast and slow: On dual process theory and metacognition in embodied AI, 2020. URL https://openreview.net/forum?id=iFQJmvUect9.

Henry Prakken. *Logical tools for modelling legal argument: a study of defeasible reasoning in law*, volume 32. Springer Science & Business Media, 2013.

Rasmus Rendsvig and John Symons. Epistemic Logic. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, Summer 2021 edition, 2021.

Silvia Richter and Matthias Westphal. The lama planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research*, 39: 127–177, 2010. doi: 10.1613/jair.2972.

Idriss Riouak. Non-well-founded set based multi-agent epistemic action language. Unpublished MSc Thesis, 2019.

Leif Benjamin Rodenhäuser. *A matter of trust: Dynamic attitudes in epistemic logic*. PhD thesis, Universiteit van Amsterdam [Host], 2014.

Stuart J. Russell and Peter Norvig. *Artificial Intelligence - A Modern Approach, Third International Edition*. Pearson Education, 2010. ISBN 978-0-13-207148-2. URL `http://vig.pearsoned.com/store/product/1,1207,store-12521_isbn-0136042597,00.html`.

Boris Schling. *The Boost C++ Libraries*. XML Press, 2011. ISBN 0982219199.

Amitai Shenhav, Matthew M. Botvinick, and Jonathan D. Cohen. The expected value of control: An integrative theory of anterior cingulate cortex function. *Neuron*, 79(2):217–240, July 2013. ISSN 0896-6273. doi: 10.1016/j.neuron.2013.07.007.

Raymond R. Smullyan. *First-order logic*, volume 43. Springer-Verlag Berlin Heidelberg, 1968. ISBN 978-3-642-86718-7. doi: 10.1007/978-3-642-86718-7.

Tran Cao Son, Enrico Pontelli, Chitta Baral, and Gregory Gelfond. Finitary s5-theories. In *European Workshop on Logics in Artificial Intelligence*, pages 239–252. Springer, 2014. doi: 10.1093/jigpal/jzm059.

Bjarne Stroustrup. *The C++ Programming Language*. Addison-Wesley Professional, 4th edition, 2013. ISBN 0321563840.

Alice Tarzariol. Evolution of algorithm portfolio for solving strategies. In Alberto Casagrande and Eugenio G. Omodeo, editors, *Proceedings of the 34th Italian Conference on Computational Logic, Trieste, Italy, June 19-21, 2019*, volume 2396 of *CEUR Workshop Proceedings*, pages 327–341. CEUR-WS.org, 2019. URL `http://ceur-ws.org/Vol-2396/paper37.pdf`.

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

Alan M. Turing. Computing machinery and intelligence. *Mind*, 59(October):433–460, 1950. doi: 10.1093/mind/LIX.236.433.

Johan Van Benthem, Jan Van Eijck, and Barteld Kooi. Logics of communication and change. *Information and computation*, 204(11):1620–1662, 2006. doi: 10.1016/j.ic.2006.04.006.

Hans Van Ditmarsch, Wiebe van Der Hoek, and Barteld Kooi. *Dynamic Epistemic Logic*, volume 337. Springer Netherlands, 2007. ISBN 978-1-4020-6908-6. doi: 10.1007/978-1-4020-5839-4.

Hans van Ditmarsch, Wiebe van der Hoek, Joseph Y. Halpern, and Barteld Kooi, editors. *Handbook of Epistemic Logic*. College Publications, 2015. ISBN 978-1-84890-158-2.

Jan van Eijck. *Dynamic Epistemic Modelling*. CWI. Software Engineering [SEN], 2004.

Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA, 2009. ISBN 1441412697.

Hai Wan, Rui Yang, Liangda Fang, Yongmei Liu, and Huada Xu. A complete epistemic planner without the epistemic closed world assumption. In *IJCAI International Joint Conference on Artificial Intelligence*, pages 3257–3263, Buenos Aires, Argentina, July 25-31 2015. URL `http://ijcai.org/Abstract/15/459`.

Quan Yu, Ximing Wen, and Yongmei Liu. Multi-agent epistemic explanatory diagnosis via reasoning about actions. In Francesca Rossi, editor, *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013*, pages 1183–1190. IJCAI/AAAI, 2013. URL `http://www.aaai.org/ocs/index.php/IJCAI/IJCAI13/paper/view/6631`.