

Planning while Believing to Know



UNIVERSITY OF UDINE

Department of Mathematics, Computer Science and Physics

CANDIDATE:
Francesco Fabiano

ADVISOR:
Prof. Agostino Dovier

CO-ADVISORS:
Prof. Enrico Pontelli
Prof. Alessandro Dal Palù

Thesis submitted for the degree of
Doctor of Philosophy in Computer Science, Mathematics and Physics

Cicle: XXXIV

Years: 2018–2021

Ai miei Genitori

To my Parents

Abstract

Over the last few years, the concept of *Artificial Intelligence* (AI) has become essential in our daily life and in several working scenarios. Among the various branches of AI, *automated planning* and the study of *multi-agent systems* are central research fields. This thesis focuses on a combination of these two areas: that is, a specialized kind of planning known as *Multi-agent Epistemic Planning*. This field of research is concentrated on all those scenarios where agents, reasoning in the space of knowledge/beliefs, try to find a plan to reach a desirable state from a starting one. This requires agents able to reason about her/his and others' knowledge/beliefs and, therefore, capable of performing *epistemic reasoning*. Being aware of the information flows and the others' states of mind is, in fact, a key aspect in several planning situations. That is why developing autonomous agents, that can reason considering the perspectives of their peers, is paramount to model a variety of real-world domains.

The objective of our work is to formalize an environment where a complete characterization of the agents' knowledge/beliefs interactions and updates are possible. In particular, we achieved such a goal by defining a new action-based language for Multi-agent Epistemic Planning and implementing epistemic solvers based on it. These planners, flexible enough to reason about various domains and different nuances of knowledge/belief update, can provide a solid base for further research on epistemic reasoning or real-base applications. This is true, especially considering that one of the proposed approaches formally verifies that the obtained plan is correct with respect to semantics on which is based.

This dissertation also proposes the design of a more general epistemic planning framework. This architecture, following famous cognitive theories, tries to emulate some characteristics of the human decision-making process. In particular, we envisioned a system composed of several solving processes, each one with its own trade-off between efficiency and correctness, which are arbitrated by a *meta-cognitive* module.

Contents

List of Figures	ix
List of Tables	xi
1 Introduction & Preliminaries	1
1.1 Motivation	1
1.2 Planning: Notation and Concepts	4
1.3 Reasoning about Knowledge and Beliefs	16
1.4 Multi-agent Epistemic Planning	28
2 Possibilities-Based MEP Action Language	39
2.1 Background	39
2.2 The Epistemic Action Language $m\mathcal{A}^\rho$	55
3 Communication with Trust	65
3.1 Trust in $m\mathcal{A}^\rho$	65
3.2 Capturing Trust with Update Models	77
4 Trust, Misconception, and Lies in MEP	83
4.1 Agents' Attitudes and Inconsistent Beliefs	83
4.2 Updated Transition Function	87
4.3 Related Work	100
5 Comprehensive Multi-Agent Epistemic Planners	103
5.1 Background	103
5.2 EFP: an Epistemic Forward Planner	106
5.3 PLATO: an Epistemic Planner in ASP	137
6 "Fast and Slow" Epistemic Planning	149
6.1 Background	149
6.2 MEP System-1 and System-2	153
6.3 A Fast and Slow Epistemic Architecture	159
7 Conclusion	167

Appendices

A Propositions Proofs	171
A.1 Preliminary Definitions	171
A.2 Proofs of Propositions 2.3 to 2.5	174
A.3 Proofs of Propositions 3.1 and 3.2	179
A.4 Proof of Proposition 4.1	186
A.5 Proofs of Propositions 5.1 to 5.3	192
Bibliography	201

List of Figures

1.1	The World Block domain.	6
1.2	The Planning Problem in the World Block domain.	8
1.3	A planning tree for the World Block domain.	11
1.4	The Vacuum-Cleaner domain with both dirty rooms.	13
1.5	The execution of the action <code>Clean</code> in a conformant domain.	14
1.6	Vacuum-Cleaner domain AND-OR tree.	14
1.7	The Soccer domain.	16
1.8	The Kripke structure that represents Planning Domain 1.4.	24
1.9	The Kripke structure of the Planning Domain 1.4 variation.	30
1.10	Example of e-State after update after an announcement.	31
1.11	The execution of the plan $\langle \text{open}\langle A \rangle, \text{peek}\langle A \rangle, \text{announce}\langle A \rangle(\text{heads}) \rangle$. 35	
2.1	Execution of an action instance.	43
2.2	Update templates of action types described by Baral et al. [2022].	48
2.3	Well-founded sets represented through graphs [Aczel, 1988].	49
2.4	Representation of the non-well-founded set $\Omega = \{\Omega\}$ [Aczel, 1988].	50
2.5	Bisimilar Kripke structures.	52
2.6	Representation of a generic possibility w	54
2.7	From a possibility to a Kripke structure.	55
2.8	The initial state.	62
2.9	Execution of <code>distract_C</code> $\langle A \rangle$	62
2.10	Execution of <code>open</code> $\langle A \rangle$	63
2.11	Execution of <code>peek</code> $\langle A \rangle$	63
2.12	e-State, generated by $m\mathcal{A}^\rho$ and $m\mathcal{A}^*$, size comparison.	64
3.1	The initial e-state described in Planning Domain 3.1.	71
3.2	The result of applying an <i>un</i> -trustworthy announcement.	71
3.3	The result of applying a <i>mis</i> -trustworthy announcement.	74
3.4	The update template (Σ, σ) for the <i>un</i> -trustworthy announcement.	79
3.5	The update template (Σ, σ) for the <i>mis</i> -trustworthy announcement.	81
4.1	The initial state of Planning Domain 4.1.	93
4.2	Correct sensing example.	94

4.3	Wrong sensing example.	95
4.4	Announcement with trustful & mistrustful listeners example.	96
4.5	Announcement with mistrustful & stubborn listeners example.	97
4.6	Lie example.	98
5.1	Comparison between EFP 1.0 and P-MAR on SC	112
5.2	Example of a planning graph.	130
6.1	The schema of MC-2	158

List of Tables

1.1	Knowledge and beliefs axioms [Fagin et al., 1995, chapter 3].	26
1.2	SAT problem complexity for DEL [Fagin et al., 1995].	37
1.3	Complexity of the <i>plan existence problem</i> [Bolander et al., 2015].	38
2.1	Action types and observability relations Baral et al. [2015].	46
2.2	Observability relations of the actions instances in Δ	61
5.1	Runtimes for the CC domain for EFP 1.0 and EFP 2.0.	112
5.2	Runtimes for the GR domain for EFP 1.0 and EFP 2.0.	113
5.3	Runtimes for the CB domain for EFP 1.0 and EFP 2.0.	113
5.4	Runtimes for the AL domain for EFP 1.0 and EFP 2.0.	114
5.5	e-States' size comparison between different solving processes.	115
5.6	Runtimes for the GR domain for EFP 2.0 and RP-MEP.	117
5.7	Time consumption of EFP 2.0 and EFP 2.1 on the CB domain.	122
5.8	Time consumption of EFP 2.0 and EFP 2.1 on the AL domain.	122
5.9	Time consumption of EFP 2.0 and EFP 2.1 on the GR domain.	122
5.10	Time consumption of EFP 2.0 and EFP 2.1 on the CC domain.	123
5.11	Memory consumption of EFP 2.0 and EFP 2.1 on the CB domain.	123
5.12	Memory consumption of EFP 2.0 and EFP 2.1 on the SC domain.	123
5.13	Memory consumption of EFP 2.0 and EFP 2.1 on the GR domain.	124
5.14	Memory consumption of EFP 2.0 and EFP 2.1 on the CC domain.	124
5.15	Solving times of the uninformed searches of EFP 2.1 on CB	125
5.16	Solving times of the uninformed searches of EFP 2.1 on AL	126
5.17	Solving times of the uninformed searches of EFP 2.1 on SC	126
5.18	Solving times of the uninformed searches of EFP 2.1 on GR	126
5.19	Solving times of the uninformed searches of EFP 2.1 on CC	127
5.20	Comparison of uninformed and informed search on the CC domain.	128
5.21	Performances comparison between EFP 2.1 and PLATO.	146

Thesis Organization

Here we will provide a high-level overview of each chapter of the thesis. Alongside the chapters' content, we will also provide references to published scientific articles—produced during the Ph.D. period—that constitute the backbone of this work. These articles have been validated and enriched by the peer-review process, providing a solid foundation for this dissertation.

1. The first chapter serves as an introduction to the whole thesis. It starts by providing the motivation for our work, explaining why we decided to direct our research efforts to *Multi-agent Epistemic Planning*. It then illustrates some basic notions related: *(i)* to the planning area; *(ii)* to the epistemology world; and *(iii)* to their connections. While this chapter does not present any new contributions, it helps in setting the foundations necessary to understand the actual contributions. Given the introductory nature of this chapter, we decided to take inspiration from more experienced authors. In particular, the prominent reference for the planning introduction was provided by Russell and Norvig [2010] while, for the epistemic preamble we referred to Fagin et al. [1995], Bolander and Andersen [2011], Baral et al. [2015], van Ditmarsch et al. [2015].
2. The second chapter introduces the first contribution of our research. In particular, we present a variation of an epistemic action language, *i.e.*, a language used to define Multi-agent Epistemic Planning problems. This new language is based on non-well-founded data structures, called *possibilities*—initially theorized by Gerbrandy and Groeneveld [1997]—rather than the more classical Kripke structures. The chapter illustrates the components of the language highlighting its advantages with respect to its predecessor, concluding with some remarks on the correctness of the new specification. To provide enough information about the newly adopted possibilities, we mostly referred to works by their original authors, *i.e.*, Gerbrandy and Groeneveld [1997], Gerbrandy [1999]. On the other hand, the formal definition of the new language was firstly presented in our work Fabiano et al. [2019] and later improved in Fabiano et al. [2020].
3. Chapter three further analyzes the concept of epistemic action languages. In particular, it explores how to enrich the aforementioned language with features

that would allow it to represent more realistic scenarios. We accomplished that by formalizing the idea of *trust* between agents. This extension has been devised for both the possibilities-based and the Kripke structure-based languages. Finally, we provide some fundamental properties to ensure that the communications with trust behave as expected. This chapter derives from the work Fabiano [2020] where we initially tackled the idea of formalizing trust in our epistemic action language.

4. As a final advancement in our formalization of a general epistemic language specification we present, in chapter four, the idea of *agents' attitudes*. These attitudes are used to associate each agent with a particular set of biases about the information received by others. With these extra characterizations, we can formalize domains with a wider spectrum of interactions; allowing for epistemic planning domains to become even more realistic. As for the previous chapters, we captured some fundamental properties of this new addition ensuring its correctness. This idea was firstly explored and formalized in Fabiano et al. [2021a].
5. In the fifth chapter we present the implementation of a *general and comprehensive epistemic solver*. This C++ planner, called EFP, integrates all the previous theoretical advancements and constitutes a tool that we hope will be adopted by the community as the basis for future research. EFP is able to plan while considering belief relations and concepts such as lies, misconceptions, trust, and so on. While generality is our primary concern, EFP shows state-of-the-art performances in reasoning on complete epistemic states as we can see from the various experimental evaluations presented in the chapter. Finally, we also present PLATO, a version of the planner in Answer Set Programming. This planner and its use of the declarative approach are then compared with EFP and its more classical imperative paradigm. Thanks to its declarative nature PLATO allows us to formally validate its behaviour, with respect to the underlying semantics, and therefore the computed plans. Furthermore, this permits to empirically confirm the results obtained by the versions of EFP that implement the underlying action language of PLATO.

The EFP version presented is the result of several scientific productions where its internal structure has been optimized and enriched; in chronological order, these are Le et al. [2018], Fabiano [2019], Fabiano et al. [2020, 2021a]. PLATO, instead, has been formalized and implemented initially in Burigana et al. [2020].

6. Chapter six discusses the integration of a famous cognitive theory, that is “thinking fast and slow” by Kahneman [2011], into the modern concept of AI.

This chapter stems from a collaboration with a research group from the IBM Thomas J. Watson Research Center. While the joint project aims to analyze cognitive theories to widen the AI capabilities, in this chapter we focus on how this research can affect the epistemic planning setting. In particular, the chapter will identify what it means to think *fast* and *slow* in Multi-agent Epistemic Planning, examining also the role of the meta-cognition in an architecture that employs the aforementioned paradigm. The chapter also introduces an architecture that, following the schema proposed by Kahneman, is able to tackle epistemic planning problems. This final contribution is based, alongside countless hours of discussion with the IBM research group, on the scientific contributions by Booch et al. [2021], Fabiano et al. [2021b], Ganapini et al. [2021, 2022].

7. The last chapter concludes the thesis with some final remarks on the various contributions and with a brief description of possible future works.

An investment in knowledge pays the best interest.

— Benjamin Franklin
Poor Richard's Almanac
[Franklin, 1750]

1

Introduction & Preliminaries

Contents

1.1	Motivation	1
1.2	Planning: Notation and Concepts	4
1.2.1	Basic Concepts	5
1.2.2	Planning Problem Categories	9
1.3	Reasoning about Knowledge and Beliefs	16
1.3.1	Epistemic Logic	18
1.4	Multi-agent Epistemic Planning	28
1.4.1	Epistemic Actions	29
1.4.2	Multi-agent Epistemic Planning Problem	32
1.4.3	Complexity Overview	35

1.1 Motivation

Artificial Intelligence (AI for short) is a term, coined by McCarthy, Minsky, Rochester, and Shannon in 1955, used to capture the idea of autonomous machines which have capabilities that allow them “to think”. While exploring what it means “to think” deserves a dissertation on its own, we make use of this term in a loose and non-formal way to provide the reader with a general intuition of what an autonomous agent should accomplish. This concept stems from one of the most important scientific figures of the last century, considered to be the founding father

of computer science and Artificial Intelligence, Alan Turing. In the first sentence of his publication “*Computing Machinery and Intelligence*” [Turing, 1950], Turing proposed “*to consider the question, ‘Can machines think?’*”. He also provided meaning to what it means for a machine “to think” introducing the well-known *Turing’s test*, which states that machines can be considered intelligent when they can mimic the behavior of humans.

With his contribution, Turing effectively initiated the scientific quest of formalizing and constructing agents that are able to act on the world out of their own volition. After McCarthy, Minsky, Rochester, and Shannon proposed and organized the first meeting on Artificial Intelligence in 1956 [McCarthy et al., 2006], researchers started to investigate this topic with several revolutionary accomplishments in both theoretical and practical aspects of AI. In particular, the idea of intelligence has been refined to incorporate the fundamental aspect of *rationality* that does not always (nor often) coincide with human behavior. That is why, nowadays, the idea of machines’ intelligence is not limited to the sole human behavior imitation but also considers the possibility of agents that reason, or act, following logic, as elegantly summarized by Russell and Norvig in their book [Russell and Norvig, 2010, chapter 1].

Alongside the novelties introduced at the conceptual level, the AI community formalized and developed several techniques that permit to model agents which can solve intricate problems in autonomy. These techniques range from the use of various formal logics, and in particular *modal logic*, to the creation of neural-based structures. The former is an area of study that stems from the field of philosophy which, after the initial efforts of Clarence Irving Lewis, evolved rapidly [Ballarin, 2021] and has become an essential tool to define rational behavior for our systems. The latter is a technology that, imitating the physiology of our brain, allows the agents to perform reasoning tasks emulating (to some degree) the human behavior. While this idea was firstly studied in the early days of AI, only recently, thanks to figures such as Geoffrey Hinton, Yoshua Bengio, and Yann LeCun, *neural networks* are adopted to solve a wide spectrum of problems, as reported by Bengio et al. [2021].

While the field of AI has constantly evolved after the intriguing question posed by Turing in 1950, over the last few years the concept of Artificial Intelligence has become more and more prominent in our life, whether we are computer science researchers or not. The concept of autonomous agents, often identified by software processes, performing tasks of different nature has been accepted and embraced in both our daily life and in the industry. That is why, AI-driven solutions are, nowadays, frequently used to tackle problems that range from mundane ones—*e.g.*, teaching how to play Sudoku [Hanson, 2021]—to very intricate tasks that require a high-level of expertise—*e.g.*, analyzing CT scans to help radiologists in identifying anomalies [Chu et al., 2019, Fabiano and Dal Palù, 2022]. Not only AI techniques are widely deployed, but it is becoming essential for the majority of the real-world scenarios, *e.g.*, Industry 4.0, to exploit tools derived from the fields of automated reasoning and knowledge representations [Lasi et al., 2014].

Even if AI is gaining popularity, most of the research efforts are not directed to this topic as a whole but to specialized sub-areas; *e.g.*, *natural language recognition*, *knowledge representation/manipulation*, and *formal verification*. In particular, the field of *automated planning* is one of the most important and most studied branches of AI. As said by Russell and Norvig [2010, chapter 10], “*we have defined AI as the study of rational action, which means that planning—devising a plan of action to achieve one’s goals—is a critical part of AI*”. That is why we decided to focus our research on the planning problem and, in particular, on those situations where multiple entities interact with each other. These scenarios, known as *multi-agent* for the presence of multiple active entities, are ubiquitous in everyday life and represent the majority of the “real-world” problems.

To correctly address multi-agent problems, a solving process needs to reason not only on the state of the world but also on its information flows. As said by [Van Ditmarsch et al., 2007] “*information is something that is relative to a subject who has a certain perspective on the world, called an agent, and that is meaningful as a whole, not just loose bits and pieces. This makes us call it knowledge*

and, to a lesser extent, belief”. That is why *epistemic*¹ and *doxastic*² reasoning come into play in formalizing such scenarios. These types of automated reasoning are used to capture the knowledge or beliefs relations among multiple agents and provide a tool to formalize those settings where the information flows must be considered by the solving process, *e.g.*, economy [Aumann et al., 1995], security [Balliu et al., 2011], justice [Prakken, 2013] and politics [Carbonell Jr, 1978].

1.2 Planning: Notation and Concepts

According to Cambridge Dictionary [2021], a *plan* is “*a method for doing or achieving something, usually involving a series of actions [...]*” implying that *planning* permeates every thought-out process performed by humans, animals, or even machines. To plan is, in fact, to devise a way of reaching an objective, whether this goal is to have enough food for the day or to build a skyscraper. The ability to divide processes, independently of their complexity, into “smaller” and more manageable ones is paramount to accomplish one’s objectives and, ultimately, to manipulate the environment to her/his advantage. That is why, designing autonomous agents that incorporate the ability to select the best course of action to achieve their goals, is of the utmost importance in Artificial Intelligence.

While the concept of automated planning has several variations (*e.g.*, *classical*, *conformant*, *epistemic*, etc.) that are used to describe different real-world scenarios, all of them share the same objective: given an initial configuration of the environment, find a sequence of permitted actions to reach the desired configuration of the same environment.

Given its importance inside the AI community, the *planning problem* is a long-studied and researched topic. That is why, we will not provide a comprehensive introduction of this field addressing the interested readers to the book of Russell and Norvig [2010, chapters 10 and 11] for a much more complete and elegant description of automated planning.

¹From the ancient Greek term ‘episteme’ (ἐπιστήμη) that means ‘knowledge’.

²From the ancient Greek term ‘doxasia’ (δοξασία) that means ‘belief, opinion’.

1.2.1 Basic Concepts

In what follows we will introduce the basic terminology and concepts, related to the planning environment, that will be essential to present the contributions of this thesis. The well-known *Block World* domain, presented in Planning Domain 1.1, will be used as a running example to better explain the concepts introduced throughout this section. In particular, this domain will be used to describe the key features of planning.

Planning Domain 1.1: Block World

The Block World, due to its simplicity, is one of the most employed domains when it comes to explaining the basics of planning. This domain consists of a few simple elements:

- *blocks* of the same size that can be placed either: on the table, or on top of another block; and
- a *mechanical arm* that can move the blocks and can determine whether it is holding a block or not.

Moreover, there are some constraints that regulate the Block World:

- the mechanical arm can only hold, and therefore move, one block at the same time; and
- a block can only be placed on top of a *clear* block—a block with no blocks on top of it and that is not held by the mechanical arm—on the table.

In what follows, we will provide a series of definitions, each followed by an example based on the Block World domain, to formalize the ideas of *state*, *agent*, *action*, *planning problem*, *transition function*, and *solution* in planning.

The first fundamental concept that we need to introduce is the idea of *planning state*. This concept, formally introduced in Definition 1.1, is used to define a static “picture” of the environment expressing its properties thanks to *fluent literals*—Boolean propositional variables that can change their truth value over time—that represent different aspects of the state itself.

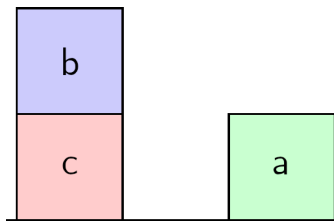


Figure 1.1: The World Block domain.

Definition 1.1: Planning State

A *state* of the domain is a configuration of the environment described by the domain, referred to as *world*, represented as a conjunction of *ground fluent literals*.

Example 1.1: Planning State Figure 1.1 represents a state in the Block World domain. This state is defined by the following positive fluent literals: $\{\text{onTable_A}, \text{onTable_C}, \text{onC_B}, \text{Clear_Arm}\}$. In this description and in what follows, the negative fluent literals, *e.g.*, $\neg\text{onTable_B}$ or $\neg\text{onA_C}$, might be omitted for the sake of readability.

Next, let us introduce the idea of *agent*. Intuitively, an agent is an entity that acts upon the domain interacting with its elements and/or with other agents to achieve her/his goal.

Definition 1.2: Planning Agent [McNeill and Bundy, 2010]

An agent is an entity that responds to goals through forming plans to achieve them and then, possibly, enacts these plans through interacting with the domain.

Example 1.2: Agent In the Block World domain an agent is the (simulated) mechanical arm that moves the blocks to find the desired configuration.

After defining the idea of agents as entities that change the states trying to reach the goal executing some actions, we need to formalize the concept of *planning action*. Let us note that execution in a software environment is just a simulation of the actions.

Definition 1.3: Action

An *action* in planning is an operation, made by some agent, that changes the actual world or its perception. Actions can have *executability conditions* that express when an action is, as the name suggests, executable and when it is not.

Example 1.3: Action Given the state in Figure 1.1 we can give an example of executable and not-executable actions.

- An executable action is **take(B)**. This action states that the mechanical arm takes block B and keeps it. The executability conditions of this action are $\{\text{ClearArm}, \neg\text{onB_A}, \neg\text{onB_C}\}$ which are respected in the current state. These conditions read as “The action **take(B)** is executable if: (i) the mechanical arm is not holding any block; (ii) block A is not on top of block B; and (iii) block C is not on top of block B.”
- An example of not-executable action is **take(C)**, which demands to mechanical arm to take block C. This action is not executable because block C is not clear. More formally, the executability conditions of this action are $\{\text{ClearArm}, \neg\text{onC_A}, \neg\text{onC_B}\}$ and, since $\neg\text{onC_B}$ is not respected, the action cannot be executed.

The *planning problem* (Definition 1.4) formally describes (i) the scenario in which $n \geq 1$ agents act upon; (ii) the starting point; and (iii) the desired configuration. The combination of these descriptions, alongside the formalization of how an action affects the world (Definition 1.5), is what allows the agents to find the *plan* (Definition 1.6).

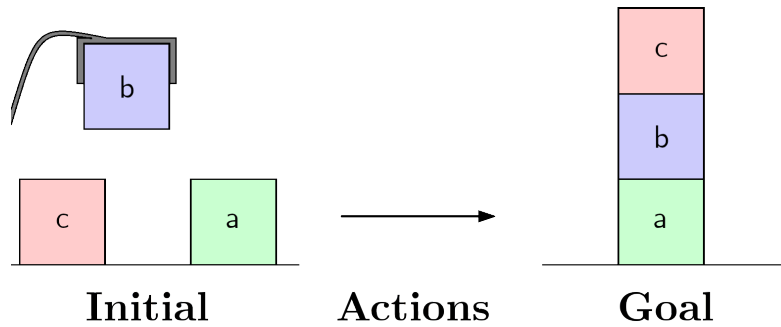


Figure 1.2: The Planning Problem in the World Block domain.

Definition 1.4: Planning Problem

A *planning problem* is a tuple $\langle D, I, G \rangle$ where:

- D is an *action domain* expressed in some language: *i.e.*, D describes the properties of interest of the environment in which the agents are acting upon and also specifies how the agents themselves can manipulate these properties through actions;
- I is a set of states of the domain—called *Initial state*—that describes the diverse (possible) starting configurations of the world. The example in Figure 1.2, comprised of single state, is described as: $\{\text{onTable_A, onTable_C, holding_B, } \neg\text{ClearArm, } \neg\text{onA_B, } \neg\text{onA_C, } \neg\text{onB_A, } \neg\text{onB_C, } \neg\text{onC_A, } \neg\text{onC_B, } \neg\text{holding_A, } \neg\text{holding_C}\}$.
- G is a set of states of the domain—called *Goal state*—that describes the desired configurations of the domain. The example in Figure 1.2, once again composed of a single state, is described as: $\{\text{onTable_A, onA_B, onB_C, ClearArm}\}$.

Before introducing the concept of transition function let us note that since the action execution may be non-deterministic, and therefore contemplates multiple states where only one should be intuitively created, we must consider the update to be capable of handling sets of states. These sets intuitively represent all the possible states that can be reached considering the various non-deterministic effects. The special case where all the actions are deterministic simply considers these sets to be singletons.

Definition 1.5: Transition Function

A *transition function* Φ is a function that, given a starting state and an action, returns a set of states in which the world can be after the execution of the action in the starting state. More formally, $\Phi : 2^\Sigma \times A \rightarrow 2^\Sigma$ where: Σ is the set of all the possible states and A the set of all the possible actions in the domain. If an action a is not executable in a state $s \in \Sigma$ then we will have $\Phi(a, s) = \emptyset$. Finally, we consider $\Phi : \Sigma \times A \rightarrow \Sigma$ when the actions are constrained to have deterministic effects.

Definition 1.6: Plan/Solution

A *plan/solution* for the generic planning problem $\langle D, I, G \rangle$ is a sequence of actions $\in D$ that, when executed, transforms the the given initial state I into one of the desired configurations $\in G$.

Once again, assuming the actions to be constrained to have deterministic effects, a *plan/solution* is a sequence of actions $[a_1, \dots, a_n]$ such that:

- a_1 is executable in every state s belonging to I ,
- a_i is executable in every state s belonging to $\Phi_D(a_{i-1}, \dots, \Phi_D(a_1, I))$ for $i = 2, \dots, n$
- G is true in every state s belonging to $\Phi_D(a_n, \dots, \Phi_D(a_1, I))$.

Even if the presented terminology is shared across the whole planning community, as already mentioned, the research in planning is differentiated into several categories. A brief explanation of all the planning problem types, that are relevant to this thesis (except for *epistemic planning* that will have a section on its own, *i.e.*, Section 1.3), will be presented next.

1.2.2 Planning Problem Categories

Classical Planning

The idea of *classical planning* has been present since the birth of AI and it has been widely explored in the computer science community ever since. That is why, we believe that introducing this concept with an already existing description, that has been thought by far more experienced researchers, would be most appropriate. In particular, the brief yet elegant description written by Bolander and Andersen

does an excellent job in explaining what classical planning is: “*For most of its early life in the ’60s and ’70s, the field of automated planning was concerned with ways in which the problem of creating long-term plans for achieving goals could be formulated, such that solving problems of non-trivial size, would be computationally feasible. The type of planning that arose from this early work, is what is known today as Classical Planning*”.

The success of classical planning is partially due to the several restrictions imposed on the world description—*i.e.*, the domain has to be *(i) static*; *(ii) deterministic*; and *(iii) fully observable* [Ghallab et al., 2004]—that make this kind of problems more tractable and approachable. In particular, *(i)* a *static* problem is represented by a domain that is not modified by elements that are external to the domain itself. *(ii)* A domain is *deterministic* when, for each state s and action a , the transition function $\Phi(a, s)$ has at most one element—*i.e.*, there is no ambiguity in which state will be the world after the execution of the action. Determinism also implies that the plan will be in one, and only one, state at each and every step of the planning computation. *(iii)* *Fully observable* means that an agent knows the complete description of the world, that is, she/he knows the state of every fluent literal in the domain. Moreover, to maintain its simplicity the classical planning domains are, most of the time, single-agent—*i.e.*, domains where only one agent can perform the actions and has to reach the goal.

A good example of classical planning can be, once again, the World Block domain described in Planning Domain 1.1. Here the single agent—namely, the mechanical arm—knows everything about the world (if a block or itself is clear or not, for example), and every action has only one possible outcome.

Several automated tools to solve classical planning problems, known as *planners* or *solvers* (Definition 1.7), have been developed for both “scientific” and industrial purposes. These tools [Richter and Westphal, 2010, Lipovetzky and Geffner, 2014, 2017], that improve each year in terms of performance and accuracy, are the foundation of all the instruments developed by the planning community.

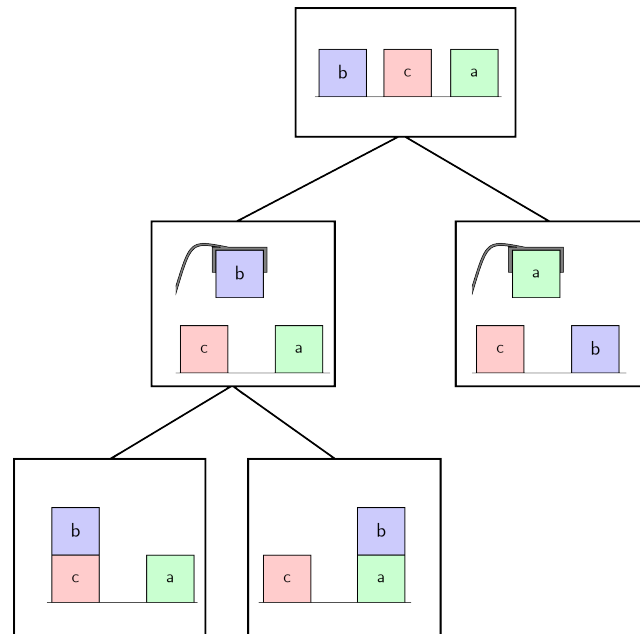


Figure 1.3: A planning tree for the World Block domain.

Definition 1.7: Planner/Solver

A *planner* (or *solver*) is a program that computes the solution of any given planning problem within a compatible domain.

During the years, to improve the performances of the various planners, different ways of representing the so-called *search-space*—an abstract representation of the paths to reachable states in a domain—have been developed. The most common and used one is referred to as *tree* and it is shown in Figure 1.3. Along with the study of how to represent the search space the planning community also studies ways to explore these spaces defining different strategies that allow the planning process to find the right balance between resources consumption and accuracy.

Classical planning is often seen as the basic form of all the other kinds of planning that, usually, consider more intricate problems or allow for a less strict description of the world. Moreover, since classical planning problems consider more constrained environments with respect to other types of planning problems, the existing solvers are the most efficient. In fact, it is not unusual to reduce, when possible, problems from more complex domains to the classical one—*i.e.*,

to elaborate the problem itself so it can be solved by a classical *planner*—even if sometimes reducing a domain may cause loss of expressiveness.

Conformant/Contingent Planning

Unfortunately, most of the real-world problems that we want to solve with planning methods do not comply with the restrictions posed by classical planners. In particular, agents may not have complete information about some properties of the world, meaning that they may not be able to retrieve the missing information until the actual execution of the plan. This type of domain is known as *conformant planning*. This “ignorance” leads to a substantial difference, with respect to classical planning, when it comes to the transition function Φ . While in classical planning applying an action produces one and only one successor, in conformant planning Φ produces a set of possible successor states. The most notable consequence is that—given that the solution for a problem must be true in all the reachable states (Definition 1.6)—the plan must be valid for all the possible configurations of the initial state.

In conformant planning the uncertainty derives from the initial state and is carried on by the actions. This means that the actions do not generate non-determinism themselves, for example through *if-else* conditions, but inherit the uncertainty from the state on which they are executed generating multiple outcomes. On the other hand, when actions do generate non-determinism we talk about *contingent* planning. To better explain this difference we will use two different descriptions of the same domain, the well-known *Vacuum-Cleaner*, introduced in Russell and Norvig [2010, chapter 2].

Planning Domain 1.2: Vacuum-Cleaner

In this domain (represented in Figure 1.4) an agent, the vacuum-cleaner, has to clean two rooms from the dirt using a sequence made from four actions **Left**, **Right**, **Clean**, **NoOperation** that do what their names suggest (described in detail in Russell and Norvig [2010, chapter 2]).

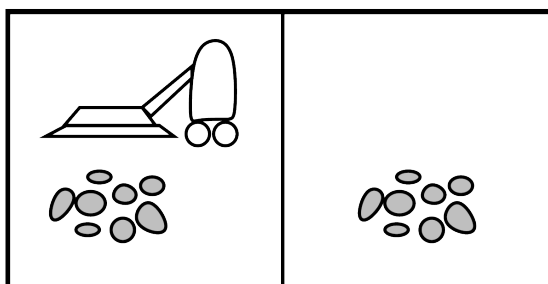


Figure 1.4: The Vacuum-Cleaner domain with both dirty rooms.

To differentiate conformant and contingent planning let us define two slightly different versions of the domain presented in Planning Domain 1.2.

The first one, used for the example of conformant planning, uses the normal interpretation of the actions but we assume that the vacuum-cleaner cannot perceive whether the rooms are dirty or clean. This implies that all the states where the agent is in the left room (accordingly to Figure 1.4) are possible initial states and the successor states of the action `Clean` are shown in Figure 1.5. A solution for this problem is $\langle \text{Clean}, \text{Right}, \text{Clean} \rangle$; this sequence of actions reaches the goal from every possible initial state.

On the other hand, the variation devised to present contingent planning uses a modification of the action `Clean`. In particular, whenever this action is applied to a room with dirt in it, the vacuum-cleaner effectively cleans the room but, sometimes, cleans the other room too. On the other hand, when the same action is applied to a clean room it sometimes deposits dirt on the carpet. A plan for the initial state, as described in Figure 1.4, is described by the *AND-OR tree* in Figure 1.6—a special data structure that is used to express the search-space in contingent planning. Without going into detail, we can see that the action `Clean` forms the so-called *OR nodes* (the ones with exiting arrows connected by an edge) that intuitively capture the idea of non-determinism.

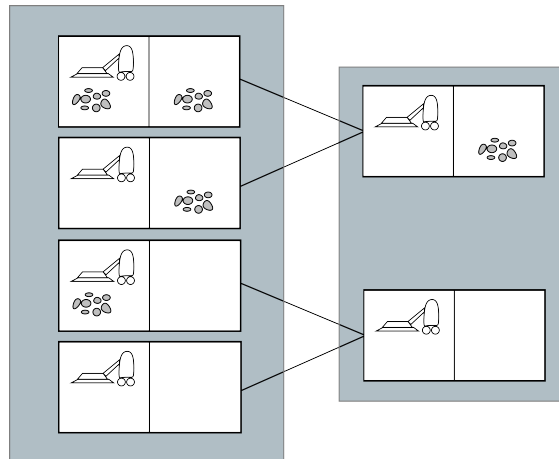


Figure 1.5: The execution of the action *Clean* in a conformant domain.

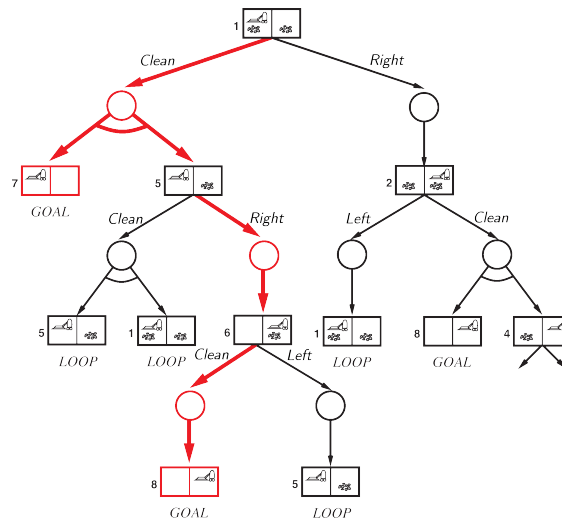


Figure 1.6: AND-OR tree for the Vacuum-Cleaner domain with non-deterministic actions. In red highlighted the solution $\langle \text{Clean}, \text{Right}, \text{Clean} \rangle$.

Multi-Agent Planning

Real-world scenarios, often, require more than a single agent that can act upon the domain. This family of planning problems that considers multiple entities is called *multi-agents planning* and it is used to model all those domains where agents' interactions are fundamental. Even if more agents acting in the same domain can, initially, seem a more efficient way to solve problems—for example, when we envision multi-agent as a means of parallelization—in reality, most of the time, having multiple entities increases the inherent complexity of the solving process.

The family of multi-agent problems engulfs several configurations of domains

that demand multiple acting entities. Next, we will list briefly some of the most known and studied configurations in the literature to provide the reader with an idea of the type of problems tackled by the multi-agent community.

The first way of classifying planning problems derives from the *agents-goal* relations. A basic subdivision is given by Bowling et al. [2005]:

- *Not-deterministic*: Each agent doesn't know which action, nor when, the other agents will perform. This implies that agents cannot accurately predict in which state the world will be in the future.
- *Cooperative*: The agents try to cooperate to reach the same goal (Planning Domain 1.3).
- *Adversary*: Under this specification, we find the most known multi-agent scenarios; *i.e.*, competitive *games*. Agents might have, in fact, opposite goals and try to reach theirs penalizing the others.
- *Overlapping Goals*: The agents just happen, without willing it, to help each other to reach their own goal.

Another distinction is based on the type of communications between agents. We can simplify the subdivision described by Fornara [2003], Katewa [2017] in two different categories of communication:

- *Free*: The agents are allowed to freely share their knowledge about the world (Planning Domain 1.3).
- *Privacy limited*: Agents can share only certain information with others. It is also possible that some agents get to share specific types of information with only a subset of the other agents.

Finally, another distinction, based on the solving process, is introduced by De Weerd and Clement [2009] and Fornara [2003]. A planning system can therefore be:

- *Centralized*: A *master* agent coordinates the action of the others.
- *Decentralized*: Each agent acts independently (Planning Domain 1.3).

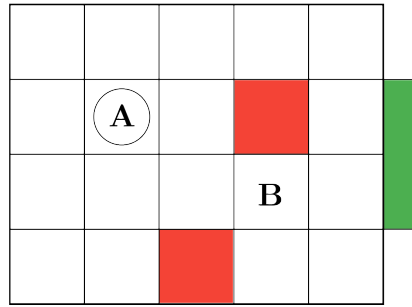


Figure 1.7: The Soccer domain.

Planning Domain 1.3: Soccer

An example of a *decentralized* multi-agent planning domain with *cooperative agents* and *without restrictions* about sharing information is a variation of the *Soccer domain* presented by Littman [1994].

In this game, each agent is placed in a single cell of the grid (Figure 1.7) and can move following the compass directions (N, S, E, and W) or *wait*. Agent **A** starts conventionally with the ball but whenever an agent tries to move in an already occupied cell she/he has to “pass” the ball to the agent that occupies that cell. The goal of this domain is to bring the ball inside the goal zone (green in Figure 1.7) without hitting the obstacles (red in Figure 1.7) in the fastest way possible.

1.3 Reasoning about Knowledge and Beliefs

Logicians have always been interested in describing *the state of the world* through formalism that would allow reasoning on the world with logic. This interest has led, among other things, to the formalization of the aforementioned planning problem and the introduction of several *modal logics* [Smullyan, 1968, Chagrov and Zakharyashev, 1997, Van Ditmarsch et al., 2007] used to describe different types of scenarios. The difference between these logics is not merely syntactical, rather it carries implications in both expressiveness and complexity. Let us take for example, without going into details, the *Boolean propositional logic* and the *linear temporal logic* (LTL). The first one, being one of the simplest logic, is mostly used to encode the world as a set of facts that can be true or not and, therefore, allows to “reduce” properties of the domain to *Boolean formulae*. The latter instead, even if it is based on propositional logic, introduces *modal operators* that allow reasoning

about time (with a little abuse of the term). The absence of these operators in the first one makes propositional logic, adopted to represent problems in the complexity class NP such as *SAT*, not expressive enough to encode problems that LTL can deal with. So, in general, we have that different logics have diverse operators and therefore are suitable for different kinds of automated reasoning.

Nevertheless, even if different, the two logics introduced above are limited to reason only on the state of the world—*i.e.*, on its “physical” properties and their changes—and since this thesis aims to tackle the planning problem while considering the beliefs of the agents, it is clear that neither propositional logic nor LTL suffice to formalize the domains that we want to explore. *Epistemic Logic*, on the other hand, is used to reason not only on the state of the world but also on the *agents’ knowledge* about the world or the others’ knowledge. Similarly, the logic that addresses the problem of reasoning on the *agents’ beliefs*—on both the physical world and on the others’ beliefs—is referred to as *Doxastic Logic* [Meyer, 2003]. The idea behind epistemic and doxastic logic is, therefore, to have a formalization that allows to reason on domains where, not only the state world is taken into consideration, but also the knowledge/beliefs that the agents have about the world and about the knowledge/beliefs of each other are considered. That is why we used these logics as the foundation for our research.

In what follows we will briefly describe the fundamental concepts that are shared between epistemic and doxastic logic. This introduction is not to be intended as a complete survey of the vast area of epistemic and doxastic logic but, rather, as a way of presenting concepts that are paramount to illustrate the contributions of this thesis. During this work, for the sake of readability, we will make several (intuitive) assumptions to avoid the need to investigate “mind-twisting” aspects of epistemology that would complicate the design of autonomous agents—*e.g.*, we assume that the agents are perfect logicians. For a far more complete, compelling and informative introduction on this topic we refer the reader to Fagin et al. [1995], Van Ditmarsch et al. [2007], van Ditmarsch et al. [2015], Rendsvig and Symons [2021].

For the sake of readability let us identify both epistemic and doxastic logic with the term “epistemic logic” when there is no need to differentiate between them (differences in reasoning on knowledge or beliefs will be further explored later in this section). Moreover, for brevity, we will make use of the term “belief” to encapsulate both the notions of an agent’s knowledge and beliefs about some information when the context permits it.

1.3.1 Epistemic Logic

Epistemology is the field of study that is concerned about knowledge and beliefs. Since its early days, Philosophy has always been intertwined with the concepts of knowledge and beliefs given that they play a central role in the development of cognitive theories as well as in the understanding of the human reasoning processes. While Aristotle is considered, among other things, to have initiated the discussion on epistemology [Rendsvig and Symons, 2021], the first logic formalization of this field is attributed to Ralph Strode, in 1387 [Boh, 1993]. This formalization, refined over the years, is what led modern philosophers in the fifties and sixties to define a complete axiomatization of the logic of knowledge and beliefs that resulted, in 1962, in the book “*Knowledge and Belief: An Introduction to the Logic of the Two Notions*” by Hintikka [1962].

While this formalization stems from the area of Philosophy, its application—*i.e.*, formally representing knowledge and/or beliefs—rapidly captured the interest of researchers of diverse areas. Notably, in the 1990s the computer science community started to embrace the idea of “reasoning about knowledge” and devised several ways to employ epistemic logic to model scenarios where autonomous agents could analyze knowledge/belief relations to, for example, better assess winning strategies. In particular, this thesis explores the interplay between (dynamic) epistemic logic and the field of planning—the so-called *Multi-agent Epistemic Planning problem*—inheriting its research scope from one of the most important works on this topic, *i.e.* “*Reasoning About Knowledge*” by Fagin et al. [1995].

Let us now introduce epistemic logic, namely the logic that allows to reason on the agents' knowledge/beliefs in static domains. In what follows we will make use of a simple instance of the *Coin in the Box domain* as a running example to present the main concepts of epistemic logic.

Planning Domain 1.4: Coin in the Box (Simplified)

Three agents, **A**, **B**, and **C**, are in a room where in the middle there is a box. The box has a lock that can only be opened with a key. Inside the box, there is a coin that lies *heads* up. In the initial configuration of this domain we have that everybody knows that:

- none of the agents know whether the coin lies heads (identified by **heads**) or tails (identified by the negation of **heads**, *i.e.*, \neg **heads**) up;
- the box is locked (identified by the negation of **opened**, *i.e.*, \neg **opened**); and
- only agent **A** has the key (identified by **haskey_A** and \neg **haskey_B**, \neg **haskey_C**).

In Planning Domain 1.4 we are presenting an example of an automated planning environment, and therefore, we should also specify the possible actions and the desired goals. Nevertheless, since we will use this example to better explain concepts of epistemic logic (which refers to static domains), for the moment we will not add any other specification to avoid unnecessary clutter. The ideas of action, transition function, and plan are, in fact, directly derived by the interaction between the field of planning and *Dynamic Epistemic Logic*. Since this combination is of great interest for our work, it will be the subject of a dedicated section, *i.e.*, Section 1.4.

Epistemic Logic Terminology

Let us consider a set \mathcal{AG} of $n \geq 1$ agents and let \mathcal{F} be a set of $m \geq 1$ propositional variables, *i.e.*, the fluent literals. With the term *epistemic world*, or simply *world* (Definition 1.8), we identify a subset of elements of \mathcal{F} —intuitively, only those that are *true* in that world. This means that a world describes a certain configuration of the environment identifying which properties hold (and, consequently, which do not). Furthermore, we use the term *pointed world* or *real world* to identify

the set of fluent literals that represents the actual configuration of the domain we are reasoning on.

Definition 1.8: Epistemic World

An *epistemic world* w is a set of propositional variables of \mathcal{F} ($w \subseteq \mathcal{F}$) which are interpreted as *true* in w ($\forall f \in w, f \models_w \top$; where \top indicates *true*). The remaining elements of \mathcal{F} , *i.e.*, the ones that are not in w , are considered to be false in w ($\forall f \in \mathcal{F} \setminus w, f \models_w \perp$; where \perp indicates *false*).

Example 1.4: Epistemic World The description of the real world of Planning Domain 1.4 is expressed by the following set of true fluent literals: $\{\text{heads}, \text{haskey_A}\}$. The remaining fluent literals, *i.e.*, opened , haskey_B , and haskey_C are considered false.

During this thesis, we will, sometimes, make use of the more “complete” representation that explicitly presents both positive and negative (preceded by the symbol \neg) fluents to strengthen the clarity of the presentation. Following this schema, the world taken into consideration would be represented as $\{\text{heads}, \text{haskey_A}, \neg\text{opened}, \neg\text{haskey_B}, \neg\text{haskey_C}\}$.

In epistemic logic, as already said, we are not only concerned with the environment properties and that is why each agent $i \in \mathcal{AG}$ is associated with an epistemic modal operator \mathbf{B}_i . This operator, intuitively, represents the beliefs of the agent i . While the operator \mathbf{B}_i captures the direct beliefs of i , we also consider the *group operator* \mathbf{C}_α . \mathbf{C}_α represents the *common beliefs* of a group of agents $\alpha \subseteq \mathcal{AG}$, *i.e.*, every agent in α believes a fact and believes that the others believe it too. The operators \mathbf{B}_i and \mathbf{C}_α allow to “enrich” the traditional definition of a *fluent formula* (Definition 1.9) and obtain the concept of *belief formula* (Definition 1.10). Several other operators, not considered by this thesis, that delineate far more complex beliefs relations—*e.g.*, the *Only Knowing* operator presented by Gerhard and Hector J. [2015]—have been devised.

Definition 1.9: Fluent Formula [Baral et al., 2015]

A *fluent formula* is a propositional formula built using the propositional variables in \mathcal{F} and the traditional propositional operators $\wedge, \vee, \Rightarrow, \neg$. A *fluent atom* is a formula composed of just an element $f \in \mathcal{F}$, instead a *fluent literal* is either a fluent atom $f \in \mathcal{F}$ or its negation $\neg f$. During this work, we will refer to fluent literals simply as *fluents*.

Definition 1.10: Belief Formula [Baral et al., 2015]

A *belief formula* is defined as follow:

- A fluent formula (Definition 1.9) is a belief formula;
- let φ be belief formula and $i \in \mathcal{AG}$, then $\mathbf{B}_i(\varphi)$ is a belief formula;
- let φ_1, φ_2 , and φ_3 be belief formulae, then $\neg\varphi_3$ and $\varphi_1 \wedge \varphi_2$ are belief formulae (the connective \vee is derived as a combination of \neg and \wedge);
- the formulae of the form $\mathbf{C}_\alpha\varphi$ are belief formulae, where φ is itself a belief formula and $\emptyset \neq \alpha \subseteq \mathcal{AG}$.

The language $\mathcal{L}_{\mathcal{AG}}^{\mathbf{C}}$ of well-formed belief formulae with *common belief*, over the sets \mathcal{F} and \mathcal{AG} , can be defined compactly way by:

$$\varphi ::= f \mid \neg\varphi \mid \varphi \wedge \varphi \mid \mathbf{B}_i(\varphi) \mid \mathbf{C}_\alpha(\varphi),$$

where $f \in \mathcal{F}$, $i \in \mathcal{AG}$ and $\emptyset \neq \alpha \subseteq \mathcal{AG}$. We read the formula $\mathbf{B}_i(\varphi)$ as “agent i believes that φ ” and $\mathbf{C}_\alpha(\varphi)$ as “it is common belief between the agents in α that φ ”. In what follows, we will simply talk about “formulae” instead of “belief formulae”, whenever there is no risk of confusion.

Example 1.5: Belief Formulae Considering Planning Domain 1.4, we can express “agent B believes that agent A has the key” with $\mathbf{B}_B(\mathbf{B}_A(\text{haskey_A}))$ and “it is common belief (between all the agents) that the box is closed” with $\mathbf{C}_{\{A,B,C\}}(\neg\text{opened})$.

Finally, from the ideas of “world” and agents’ beliefs, we can informally define a state in epistemic logic, *i.e.*, an *e-state*.

Definition 1.11: Epistemic State (e-State)

An *epistemic state* is a collection of *epistemic worlds* believed to be possible by some agent in the domain. Moreover, an epistemic state captures the agents' beliefs about both the “physical properties” and others' beliefs.

Example 1.6: Epistemic State (e-State) The *e-state* that encapsulates Planning Domain 1.4, is made of two worlds. The first, the pointed one, is the one expressed in Example 1.4 and is described as: `{heads, haskey_A}`, while the latter is identified by `{haskey_A}`. These two worlds are considered possible by all the agents (A, B, and C) as they are not able to distinguish between the case in which the coin is heads or tails up. Nonetheless, no world that contains `opened` is found in the e-state as this property is known to be false by all the agents.

Let us note that Definition 1.11 does not clearly state how the agents' beliefs are represented. To do so we will need a much more formal definition of e-state that will be provided in the next paragraph.

Epistemic Logic Semantic

In the previous paragraph, we introduced the main concepts that are involved in epistemic logic, providing for them loose and intuitive meanings. Nevertheless, if we want to adopt these notions to define autonomous reasoners we must provide formal semantics for the proposed language, supporting the ideas introduced above. In particular, in this chapter we will explore the *Kripke structures* [Kripke, 1963], a data structure widely used in literature (for instance in Fagin et al. [1995], Van Ditmarsch et al. [2007], Baral et al. [2022]) to model the semantics of epistemic logic. These structures will allow us to provide a formal meaning for: the aforementioned idea of “world”; the concept of *epistemic state* (e-state); and for the entailment of belief formulae.

Definition 1.12: Kripke Structure [Kripke, 1963]

A *Kripke structure* (Figure 1.8) is a tuple $\langle \mathcal{W}, \pi, \mathcal{B}_1, \dots, \mathcal{B}_n \rangle$, where:

- \mathcal{W} is a set of *worlds*,
- $\pi : \mathcal{W} \mapsto 2^{\mathcal{F}}$ is a function that associates an interpretation of a set of propositional variables \mathcal{F} to each element of \mathcal{W} ,
- $\mathcal{B}_i \subseteq \mathcal{W} \times \mathcal{W}$, for $i = 1, \dots, n$, is a binary relation over \mathcal{W} .

Let us observe how Definition 1.12 deals with the terminology introduced in the previous paragraphs. In fact, we have that each element of \mathcal{W} , thanks to its interpretation (described by π), identifies what we defined above as an “epistemic world”, *i.e.*, a configuration of the environment. As mentioned above, each e-world contains only the positive propositional variables, and this is true also in the worlds of a Kripke structure. For example, the e-world presented in Example 1.4, let us call it w , is identified in both cases by $w = \{\text{heads}, \text{haskey_A}\}$ (represented by the left circle in Figure 1.8).

From now on whenever we consider Kripke structures we will be referring to a small variation of the structures: the *pointed Kripke structures* (Definition 1.13) that simply add an entry point. This entry point represents what we previously called the *pointed/real world*—the actual configuration of the environment on which we are planning.

Definition 1.13: Pointed Kripke Structure

A *Pointed Kripke structure* is a pair (M, w) where $M = \langle \mathcal{W}, \pi, \mathcal{B}_1, \dots, \mathcal{B}_n \rangle$ is a Kripke structure and $w \in \mathcal{W}$. In a pointed Kripke structure (M, w) , we refer to w as the *pointed* (or *real*) *world* (represented by the bold circle in Figure 1.8).

For the sake of readability, we will make use of $M[\mathcal{W}]$, $M[\pi]$, $M[i]$ and $M[\mathcal{B}]$ to denote the components \mathcal{W} , π , \mathcal{B}_i and $\mathcal{B} = \{M[\mathcal{B}_i] \mid 1 \leq i \leq n\}$ of M , respectively. We write $M[\pi](w)$ to denote the interpretation associated to the world w via π and $M[\pi](w)(\phi)$ to denote the truth value of a fluent formula ϕ with respect to the interpretation $M[\pi](w)$. Moreover, we will often refer to a Kripke structure

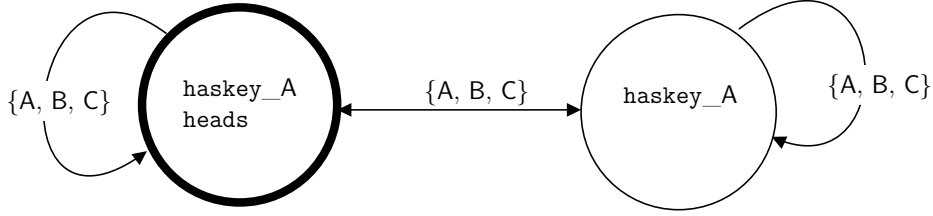


Figure 1.8: The Kripke structure that represents Planning Domain 1.4.

as a directed labeled graph, whose set of nodes is $M[\mathbf{W}]$ and whose set of edges contains (w_1, i, w_2) ³ if and only if $(w_1, w_2) \in \mathcal{B}_i$. (w_1, i, w_2) is referred to as an edge coming out of (resp. into) the world w_1 (resp. w_2).

Intuitively, a Kripke structure describes the possible worlds envisioned by the agents where the presence of multiple worlds identifies uncertainty. The relation $(w_1, w_2) \in \mathcal{B}_i$ denotes that the beliefs of agent i about the characteristics of the domain are insufficient for her/him to distinguish between the configuration described by w_1 and the one described by w_2 . This can be seen in Figure 1.8 where the two worlds are reachable from one to another by all the agents, meaning that agents A, B, and C are not able to distinguish between the worlds where the coin is heads or tails up. This results in agents' ignorance and we can say that A (and, similarly, B and C) *does not know* the coin position ($\neg \mathbf{B}_A(\text{heads}) \wedge \neg \mathbf{B}_A(\neg \text{heads})$). On the other hand, since from the pointed world agent A (and, similarly, B and C) only reaches worlds where `haskey_A` is true we can say that A believes `haskey_A` ($\mathbf{B}_A(\text{haskey_A})$). Following the informal Definition 1.11, it is clear that the information contained in a Kripke structure suffices to represent an “e-state”. In particular, the set of the possible worlds is captured by $M[\mathbf{W}]$ and the agents' beliefs can be derived by exploring the worlds' accessibility relations (starting from the real world).

More formally, in Definition 1.14, following Baral et al. [2015], we present how we can derive the truth value of belief formulae from an epistemic state representation, *i.e.*, a pointed Kripke structure. This definition allows us to provide semantics for

³ (w_1, i, w_2) denotes the edge from node w_1 to node w_2 , labeled with i .

the epistemic modal operators \mathbf{B}_i and \mathbf{C}_α , where $i \in \mathcal{AG} \supseteq \alpha$. To better express the semantics of the operator \mathbf{C}_α we will make use of an additional operator \mathbf{E}_α . While \mathbf{E}_α does not add any expressiveness to the language it allows us to express the idea of common belief more elegantly. In fact, iterating on $\mathbf{E}_\alpha^k \varphi$ easily encodes the intuitive meaning of $\mathbf{C}_\alpha(\varphi)$; that is the conjunction of the following belief formulae: (i) every agent in α knows φ ; (ii) every agent in α knows that every agent in α knows φ ; (iii) and so on *ad infinitum*.

Definition 1.14: Entailment w.r.t. a Kripke structure

Given, a fluent \mathbf{f} , the belief formulae $\varphi, \varphi_1, \varphi_2$, an agent i , a group of agents α , and a pointed Kripke structure $(M = \langle \mathcal{W}, \pi, \mathcal{B}_1, \dots, \mathcal{B}_n \rangle, \mathbf{w})$:

- $(M, \mathbf{w}) \models \mathbf{f}$ if $\mathbf{f} \in \pi(\mathbf{w})$ (or, alternatively, $\mathbf{f} \models_{\pi(\mathbf{w})} \top$);
- $(M, \mathbf{w}) \models \varphi$ if φ is a fluent formula and $\pi(\mathbf{w}) \models \varphi$ following the usual semantics of \neg and \wedge ;
- $(M, \mathbf{w}) \models \mathbf{B}_i(\varphi)$ if for each \mathbf{t} such that $(\mathbf{w}, \mathbf{t}) \in \mathcal{B}_i$ it holds that $(M, \mathbf{t}) \models \varphi$;
- $(M, \mathbf{w}) \models \neg\varphi$ if $(M, \mathbf{w}) \not\models \varphi$;
- $(M, \mathbf{w}) \models \varphi_1 \wedge \varphi_2$ if $(M, \mathbf{w}) \models \varphi_1$ and $(M, \mathbf{w}) \models \varphi_2$; and
- $(M, \mathbf{w}) \models \mathbf{E}_\alpha \varphi$ if $(M, \mathbf{w}) \models \mathbf{B}_i(\varphi)$ for all $i \in \alpha$;
- $(M, \mathbf{w}) \models \mathbf{C}_\alpha \varphi$ if $(M, \mathbf{w}) \models \mathbf{E}_\alpha^k \varphi$ for every $k \geq 0$, where $\mathbf{E}_\alpha^0 \varphi = \varphi$ and $\mathbf{E}_\alpha^{k+1} \varphi = \mathbf{E}_\alpha(\mathbf{E}_\alpha^k \varphi)$.

Axioms Systems

Following the works by Hintikka [1962], Fagin et al. [1995] let us now “quickly” define an axioms system for $\mathcal{L}_{\mathcal{AG}}^{\mathbf{C}}$ —*i.e.*, the language of well-formed formulae over \mathcal{F} and \mathcal{AG} . Such axiomatization will allow us to better categorize the properties of the language in relation to the structural constraints that an epistemic state representation, *e.g.*, a Kripke structure, must respect. In particular, we will provide the description of some properties, or axioms, on the e-states’ relations—*i.e.*, the $\mathcal{B}_1, \dots, \mathcal{B}_n$ components of a pointed Kripke structure—or, more simply, its edges. These axioms, when respected, assure that the fundamental concepts of what we call *knowledge* and *beliefs* are preserved. To better understand what we are referring

Axiom	Property of \mathcal{B}
T	$\mathcal{B}_i\varphi \Rightarrow \varphi$
4	$\mathcal{B}_i\varphi \Rightarrow \mathcal{B}_i\mathcal{B}_i\varphi$
5	$\neg\mathcal{B}_i\varphi \Rightarrow \mathcal{B}_i\neg\mathcal{B}_i\varphi$
D	$\neg\mathcal{B}_i\perp$
K	$(\mathcal{B}_i\varphi \wedge \mathcal{B}_i(\varphi \Rightarrow \psi)) \Rightarrow \mathcal{B}_i\psi$

Table 1.1: Knowledge and beliefs axioms [Fagin et al., 1995, chapter 3].

to, let us provide both the name and the formal definition of these axioms (the Left and Right column of Table 1.1, respectively).

Now we will give a brief description of the five axioms that we introduced in Table 1.1. More details on the axioms and their properties can be found in the work by Fagin et al. [1995, chapter 3].

- **T**: Has been introduced to capture the difference between knowledge and belief. When this axiom holds the real world must reflect the agents' knowledge, otherwise the agents might believe something that is not true in the actual configuration of the environment.
- **4**: Models the concept of positive introspection; this means that an agent must be aware of her/his beliefs.
- **5**: Models the concept of negative introspection; similarly to **4** an agent must be aware of what she/he does not believe.
- **D**: Introduced to ensure that an agent cannot believe "False".
- **K**: Expresses that the agent's beliefs are closed under logical consequence.

From now on, with **KD45**_{*n*}-state we will indicate e-states that consider *n* agents and respect the axioms **4**, **5**, **D**, and **K**. Similarly we will refer to the e-states on *n* agents that respect all the aforementioned axioms (**T**, **4**, **5**, **D**, and **K**) as **S5**_{*n*}-state.

Knowledge or Belief

As pointed out in the previous paragraphs the modal operator \mathbf{B}_i represents $M[i]$ —the world relations in a Kripke structure—and, as expected, different relations’ properties imply different meanings for \mathbf{B}_i . In particular, in our work, we are interested in representing the knowledge or the beliefs of the agents. The problem of formalizing these two concepts has been studied in depth bringing to an accepted formalization for both [Fagin et al., 1995]. If a relation⁴ respects all the axioms presented in Table 1.1 it is called an **S5** relation and encodes the concept of knowledge, while when it respects all the axioms but **T** characterizes the concept of belief. That is, when reasoning about knowledge we must guarantee that the underlying representation is an **S5_n**-state, while when we consider beliefs we need a **KD45_n**-state. Following these characterizations, we will refer to knowledge and belief as **S5** and **KD45** logic, respectively.

Intuitively, the difference between the two logics is that an agent cannot *know* something that is not true in **S5** but she/he can *believe* it in **KD45**. While this difference may seem superficial, designing a planning system that deals with beliefs instead of knowledge requires much more attention and increases the difficulty of the solving process. In fact, a planner that reasons about knowledge can rely on a very important property that a solver that deals with beliefs cannot exploit. That is, a planning process based on knowledge can safely assume that the agents’ information is always correct. In other words, once any agent knows a property she/he will maintain her/his knowledge even if the property changes its truth value. Moreover—since agents have to know the true nature of the information that they have—agents can also exploit the **T** axiom to reason about others’ knowledge; *e.g.*, if an agent i knows that another agent j knows a property p , then i knows p . Furthermore, these properties, combined with the most commonly used epistemic actions (introduced in the next section), ensure that the agents’ information increase monotonically. This implies that, when an agent learns something, that something can never be “unlearned” by that agent.

⁴In our case the relation between the worlds of a Kripke structure.

Contrarily, when we “drop” the **T** axiom, all of these assumptions do not hold anymore. This means that, when planning with beliefs, the solving process must account for the possibility that agents may:

- believe something that is not true in the real world;
- not being aware of changes about the truth value of already believed properties;
- believe something different from other agents;
- become ignorant about certain properties;
- announce/perceive something that does not correspond to the reality; and
- derive chains of beliefs of arbitrary length that cannot be collapsed into the same information.

All of these points make a planning system that takes into consideration beliefs, more intricate than one that is based on the **S5** logic. Nevertheless, planning on **KD45** presents the opportunity to model much more realistic (and interesting) scenarios and that is why, in this thesis, we tackle the problem of planning on beliefs.

1.4 Multi-agent Epistemic Planning

As already mentioned, reasoning about actions and information has always been one of the prominent interests since the beginning of AI [Russell and Norvig, 2010]. In particular, the continuous research effort that has characterized the field of autonomous planning is what ensured its rapid evolution. The “simple” task of reasoning in the *classical planning* environments rapidly evolved into more complex problems [Torreño et al., 2014]. This evolution, dictated both by research interests and real-world needs, developed interesting families of problems that vary in multiple aspects such as: *(i)* the number of *agents*; *(ii)* the determinism of the actions; *(iii)* the agent’s communication policies; etc.

In particular, in this thesis, we are interested in the combination of the planning field and epistemic logic. While both of these research areas have been studied and

formalized since the early sixties, their combination, *i.e.*, *Multi-agent Epistemic Planning* (MEP), is a somewhat recent introduction in the Artificial Intelligence community [Van Ditmarsch et al., 2007]. Epistemic planners, differently from most of the other solvers, are not only interested in the state of the world but also in the *knowledge* or *beliefs* of the agents. This could also be viewed, as said by Gerbrandy [1999], as “*the process of reasoning on the information itself*”. It is easy to see that an efficient autonomous reasoner that could exploit both the knowledge on the world and about other agents’ information could provide an important tool in several scenarios, *e.g.*, economy, security, justice, or politics.

Nevertheless, reasoning about knowledge and beliefs is not as direct as reasoning on the “physical” state of the world. That is because expressing, for example, belief relations between agents often implies considering *nested* and *group* beliefs that are not easily extracted from the state description by a human reader. Even if several studies [Van Ditmarsch et al., 2007, Wan et al., 2015, Muise et al., 2015, Huang et al., 2017, Le et al., 2018] have been conducted on this topic, some fundamental complications remain while characterizing MEP. In particular, the inherent complexity of reasoning on beliefs is reflected in computational overhead that brings, most of the time, infeasibility to the solving process. Moreover, modeling subtle nuances of complex ideas—*e.g.*, *trust*, *lies*, *misconception*, and so on—that are necessarily present when we reason on beliefs, is a very intricate task that makes MEP even more difficult to completely grasp. These are some of the reasons why we deem it necessary to explore the field of Multi-agent Epistemic Planning.

1.4.1 Epistemic Actions

Before exploring what it means to plan on epistemic domains, let us briefly introduce the idea of *action* in epistemic logic. As said in Moss [2015], the formalization of various types of actions and, consequently of formal languages that incorporate them, is what originated the field of *Dynamic Epistemic Logic* (DEL). While DEL is not directly used in our thesis, the formalization provided by the works on this area [Fagin et al., 1995, van Eijck, 2004, Van Ditmarsch et al., 2007, Moss, 2015]

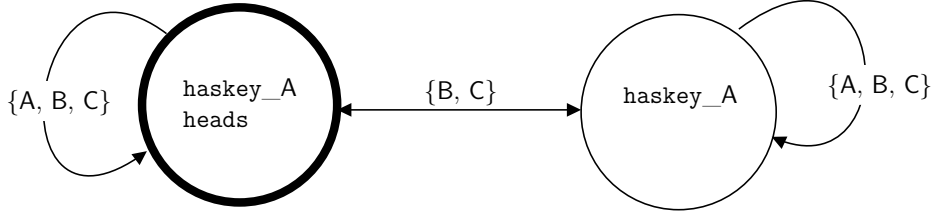


Figure 1.9: The Kripke structure that represents the Planning Domain 1.4 variation.

is what profoundly inspired our definition of *epistemic action languages* and their transition functions (presented in Chapter 2), fundamental components of our system. To better explain the concept of epistemic actions let us use a variation of Planning Domain 1.4, as a running example, where we assume that: (i) agent A believes that the coin position is *heads* (Figure 1.9); and (ii) agents B and C believe that A knows the coin position without knowing it themselves (Figure 1.9); and (iii) the agents have the ability to *announce* publicly (*i.e.*, to all the other agents) some property (*i.e.*, a fluent) of the physical world. We capture the agents’ capability with $\text{announce}\langle i \rangle(\mathbf{f})$, where $i \in \mathcal{AG}$ is an agent that executes the action⁵; and \mathbf{f} is the announced physical property. This action type is identified by the term *public announcement* and its informal semantics is: “the announcing agent tells everyone a property that she/he believes, making the other agents believe it too”. This simple semantics does not consider concepts such as lying agents, or degrees of trust. Let us note that each action description (independently from the type) is associated with an *executability condition*, that is, a belief formula that when entailed permits the action itself to be executable.

To present the public announcements formal semantics we will follow Moss [2015]. Let us note that the execution of any action implies the possible modification of the underlying e-state. In particular, to model the semantics of a public announcement, agents must believe whatever has been announced. To do so, following the notion of entailment (Definition 1.14), we must ensure that no agent can reach a world where

⁵Distinguishing between the acting agent and the others is not necessary here, but let us use this notation to be consistent with the rest of the thesis.

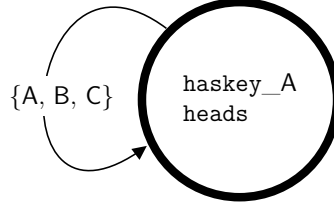


Figure 1.10: e-State of Figure 1.9 after the execution of $\text{announce}\langle A \rangle(\text{heads})$.

the announced property is false. That is, the execution of $\text{announce}\langle A \rangle(\text{heads})$ on the e-state depicted in Figure 1.9, must generate an e-state that contains only worlds that entails **heads**. This is accomplished by simply eliminating all the worlds that contain the negation of **heads** as shown in Figure 1.10. Let us note that the executability condition for this action is $\mathbf{B}_A(\text{heads})$. Following Moss [2015], the formalization of the language that also contains the aforementioned semantics for public announcements is as follows:

$$\varphi ::= \mathbf{f} \mid \neg\varphi \mid \varphi \wedge \psi \mid \mathbf{B}_i(\varphi) \mid \mathbf{C}_\alpha(\varphi) \mid [!f]\varphi,$$

where $\mathbf{f} \in \mathcal{F}$, φ is belief formula over \mathcal{AG} and \mathcal{F} , $i \in \mathcal{AG}$ and $\emptyset \neq \alpha \subseteq \mathcal{AG}$. While the epistemic operators \mathbf{B} and \mathbf{C} have already been formalized, the newly introduced operator $[!f]\varphi$ needs to be defined. Contrarily to $\mathbf{B}_i(\varphi)$ and $\mathbf{C}_\alpha(\varphi)$, that operate on a static e-state, the operator $[!f]\varphi$ must take into account also the updated version of the e-state (and that is what transforms epistemic logic into *dynamic* epistemic logic). In particular, we read the operator as “if φ is respected in the current state then, after the execution of the announcement \mathbf{f} must be believed by every agent”⁶. The axiomatization of this operator, following Moss [2015, equation 6.6], is:

$$(M, \mathbf{w}) \models [!f]\varphi \text{ iff } (M, \mathbf{w}) \not\models \varphi, \text{ or else } (M_{\mathbf{f}}, \mathbf{w}) \models \mathbf{f}.$$

where $(M_{\mathbf{f}}, \mathbf{w})$ is the epistemic state (M, \mathbf{w}) updated after the execution of the announcement.

⁶The case when an agent announces $\neg f$ is similar.

Public announcement is just one of the possible actions formalized in DEL. Since we will rely on action languages (typical of planning domains), we will make use of a formalization that is akin to the one used in the planning area. Therefore, we will not further explore DEL, addressing the interested reader to Van Ditmarsch et al. [2007], Moss [2015] for more examples of epistemic actions and a much more detailed introduction on dynamic epistemic logic.

1.4.2 Multi-agent Epistemic Planning Problem

Bolander and Andersen [2011] define epistemic planning as the generation of plans for multiple agents to achieve goals which can refer to the state of the world, the beliefs of agents, and/or the knowledge of agents. After the introduction of the classical planning problem, in the early days of Artificial Intelligence, several studies have provided the foundations for several successful approaches to automated planning. However, the main focus of these research efforts has been about reasoning within single-agent domains. In single-agent domains, reasoning about actions and change mainly involves reasoning about what is true in the world, what the agent knows about the world, how the agent can manipulate the world (using world-changing actions) to reach particular states, and how the agent (using sensing actions) can learn unknown aspects of the world.

In multi-agent domains an agent's action may not just change the world and the agent's beliefs about the world, but also may change other agents' beliefs about the world and their beliefs about other agents' beliefs. Similarly, the goals of an agent in a multi-agent world may involve manipulating the beliefs of other agents.

Although there is a large body of research on multi-agent planning Fagin et al. [1995], Durfee [2001], Bernstein et al. [2002], Guestrin et al. [2001], De Weerdts et al. [2003], Goldman and Zilberstein [2004], De Weerdts and Clement [2009], Allen and Zilberstein [2009], Muise et al. [2015], Baral et al. [2015, 2022], very few efforts address the above aspects of epistemic domains which pose several research challenges in representing and reasoning about actions and change.

Let us now formally introduce the notion of Multi-agent Epistemic Planning domain in the following definition.

Definition 1.15: MEP Domain

A *Multi-agent Epistemic Planning domain* is a tuple $D = \langle \mathcal{F}, \mathcal{AG}, \mathcal{A}, \varphi_{ini}, \varphi_{goal} \rangle$, where \mathcal{F} , \mathcal{AG} , \mathcal{A} are the sets of *fluents*, *agents*, *actions* of D , respectively; φ_{ini} and φ_{goal} are DEL formulae that must be entailed by the *initial* and *goal e-state*, respectively. The former e-state describes the domain's initial configuration while the latter encodes the desired one.

A MEP domain contains the information needed to describe a planning problem in a multi-agent epistemic setting. Given a domain D we refer to its elements through the parenthesis operator; *e.g.*, the fluent set of D will be denoted by $D(\mathcal{F})$. An *action instance* $\mathbf{a}\langle\alpha\rangle \in D(\mathcal{AI}) = D(\mathcal{A}) \times 2^{D(\mathcal{AG})}$ identifies the execution of action \mathbf{a} by a set of agents α . Multiple executors are needed in certain types of actions, for example in the so-called sensing actions (introduced in detail in the next chapters). On the other hand, actions like the public announcement introduced above, only require one executor ($|\alpha| = 1$). The *transition function* $\Phi : D(\mathcal{AI}) \times D(\mathcal{S}) \rightarrow D(\mathcal{S}) \cup \{\emptyset\}$ formalizes the semantics of action instances (the result is the empty set if the action instance is not executable). Formal definitions of this concept will be introduced in Chapters 2 to 4 where we will analyze in detail diverse transition functions. Intuitively, the features of Planning Domain 1.5 (the Planning Domain 1.4 completed with actions and goal descriptions) are:

- $\mathcal{F} = \{\text{heads}, \text{haskey_X}, \text{opened}\}$ where $X \in \mathcal{AG}$;
- $\mathcal{AG} = \{A, B, C\}$;
- $\mathcal{A} = \{\text{open}, \text{peek}, \text{announce}\}$;
- $\varphi_{ini} = \text{heads} \wedge \text{haskey_A} \wedge \mathbf{C}_{\mathcal{AG}}(\text{haskey_A}) \wedge \mathbf{C}_{\mathcal{AG}}(\neg\text{haskey_B}) \wedge \mathbf{C}_{\mathcal{AG}}(\neg\text{haskey_C}) \wedge \mathbf{C}_{\mathcal{AG}}(\neg\text{opened})$;
- $\varphi_{goal} = \mathbf{B}_A(\text{heads}) \wedge \mathbf{B}_B(\text{heads}) \wedge \mathbf{B}_C(\text{heads})$;

Planning Domain 1.5: Coin in the Box with Actions (Simplified)

Three agents, **A**, **B**, and **C**, are in a room where in the middle there is a box. The box has a lock that can only be opened with a key. Inside the box, there is a coin that lies *heads* up. In the initial configuration of this domain we have that everybody knows that:

- none of the agents know whether the coin lies heads or tails up;
- the box is locked; and
- only agent **A** has the key.

Moreover, we have that each agent can execute one of the following actions:

- **open**: an agent, if she/he has the key, can open the box. This results in all the agents believing that the box is open.
- **peek**: to learn whether the coin lies heads or tails up, an agent can peek into the box, but this requires the box to be open. This will result in the peeking agents knowing the coin position while the other agents are aware of this without knowing the coin position themselves.
- **announce**: following the public announcement semantics this will result in all the agents believing the announced coin position. As before, this action is only executable by an agent who believes the coin position to be **heads**.

Finally, the desired configuration, *i.e.*, the goal, of this instance is that all the agents (**A**, **B**, and **C**) believe that the coin is **heads** up, *i.e.*, $\mathbf{B}_A(\mathbf{heads}) \wedge \mathbf{B}_B(\mathbf{heads}) \wedge \mathbf{B}_C(\mathbf{heads})$.

The correct solution (or plan) that permits the instance presented in Planning Domain 1.5 to reach its desired goal is the sequence of action instances $\langle \mathbf{open}\langle \mathbf{A} \rangle, \mathbf{peek}\langle \mathbf{X} \rangle, \mathbf{announce}\langle \mathbf{X} \rangle(\mathbf{heads}) \rangle$ where $\mathbf{X} \in \{\mathbf{A}, \mathbf{B}, \mathbf{C}\}$. In Figure 1.11 we present the plan execution, representing the e-state resulting after the execution of each action.

We can see that in Figure 1.11b the execution of $\mathbf{open}\langle \mathbf{A} \rangle$ modified some property of the physical world, namely, the fluent **opened** became true. We will refer to this type of action, *i.e.*, the ones who modify some physical property, with the term *ontic* or, sometimes, *world-altering*. Ontic actions resemble the actions that we can find in classical planning. On the other hand, the actions that only deal with agents' beliefs, *e.g.*, **peek** and **announce**, are referred to as *epistemic actions*. We will see

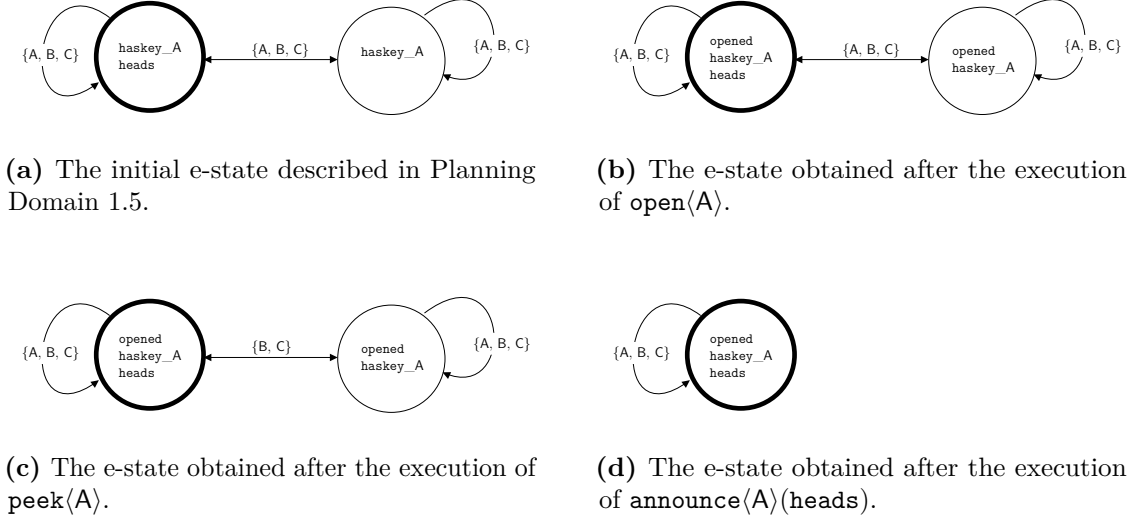


Figure 1.11: The execution of the plan $\langle \text{open}\langle A \rangle, \text{peek}\langle A \rangle, \text{announce}\langle A \rangle(\text{heads}) \rangle$.

in Chapter 2 how these two types of actions have different transition functions.

In Figure 1.11c we assume that the executor is agent A . This means that A must believe that the coin is heads up. To ensure this we simply remove all the edges that, from the pointed world, allow A to reach a world where heads is not true. We leave the edge that allows A to loop on the right world of Figure 1.11c as this is used to capture that the other agents do not know which coin position A believes to be true.

Finally, Figure 1.11d is derived following the public announcement semantics presented in Section 1.4.1. We can see how the final e-state (Figure 1.11d) respects the given goal, *i.e.*, $\mathbf{B}_A(\text{heads}) \wedge \mathbf{B}_B(\text{heads}) \wedge \mathbf{B}_C(\text{heads})$.

Let us remember that this paragraph is supposed to only provide an introduction to the field of Multi-agent Epistemic Planning. We will explore more in detail these concepts later, when we will present the contributions of this thesis.

1.4.3 Complexity Overview

Finally, as the last note on MEP, we will summarize the complexity results in the epistemic logic and in the epistemic planning fields. We will not present details on such results as we introduced them only to provide the reader with a general idea on “how hard” the problem of reasoning on information change is.

Let us start by providing some basic notions that we will use throughout this paragraph in Definitions 1.16 to 1.18.

Definition 1.16: Satisfiability of a Formula

Given a formula φ , φ is satisfiable if it is possible to find an interpretation, in our case an e-state, that makes the formula true.

Definition 1.17: Model Checking of a Formula

Given a formula φ and a model M (in our case an e-state), the model checking problem consists in determining if φ is true in M .

Definition 1.18: Plan Existence Problem

The plan existence problem consists of determining, if it exists, a solution, as defined in definition 1.6, for a planning domain D .

These definitions identify the problems of interest when planning on beliefs. Assuming that we make use of Kripke structures—other representations have the same results—to represent the e-states then: (i) Definition 1.16 is the problem of verifying whether there exists or not a Kripke structure that entails a formula; (ii) Definition 1.16 identifies the entailment of a formula over a given a Kripke structure; and (iii) Definition 1.16 represents the complete planning process. This means that identifying the complexity of these problems will allow us to characterize the MEP domain and provide us with a rough idea of how intricate is to tackle this setting. We now present a series of results that summarize the complexity of the aforementioned problems (Proposition 1.1, Tables 1.2 and 1.3). All the presented results are derived by Fagin et al. [1995], Bolander et al. [2015].

Proposition 1.1: Model Checking Complexity [Fagin et al., 1995]

There is an algorithm that, given a pointed Kripke structure (M, \mathbf{w}) , and a formula $\varphi \in \mathcal{L}_{\mathcal{AG}}^C$, determines, in POLYNOMIAL time $\mathcal{O}(|M| \times |\varphi|)$ whether $(M, \mathbf{w}) \models \varphi$, where $|M| = |M[\mathbf{W}]] + \sum_{i=0}^n |M[i]|$ with $i \in \mathcal{AG}$, and $|\varphi|$ is the number of nested operators in φ .

SAT Complexity	Epistemic logic
NP-COMplete	S5 ₁ , KD45 ₁
PSPACE-COMplete	S5 _n , KD45 _n with $n \geq 2$
EXPTIME-COMplete	S5 ^C _n , KD45 ^C _n with $n \geq 2$

Table 1.2: Complexity of the satisfiability problem with respect to the underlying Kripke structure constraints [Fagin et al., 1995].

Let us note that to analyze the *plan existence problem* we need to categorize action types into four distinct subsets. Depending on which subset an action type belongs to, the action type itself impacts differently the e-state update. This means that certain subsets of action types may increase the complexity of the plan existence problem (as we can see in Table 1.3) when taken into consideration. In MEP, as we will see in more detail in Chapter 2, the actions are often represented through graphs. These action-graphs may collapse in more “simple” data structures, *i.e.*, *singletons*, *chains*, or *trees*, for some actions and that is what makes that action part of a subset rather than another.

In particular, Bolander et al. [2015] distinguish between three different types of action-structures:

- *singletons*: that corresponds to public announcements of propositional facts;
- *chains* and *trees*: that corresponds to different types of private announcements;
- and
- *graphs*: that capture any propositional epistemic actions.

Moreover, for each one of these classes of structures, Bolander et al. analyze the complexity for the plan existence problem considering also the effects and preconditions expressive power:

- non-factual actions (changing only beliefs) with propositional preconditions;
- factual actions (changing beliefs and fluents) with propositional preconditions;
- and
- factual actions with epistemic preconditions.


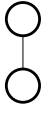
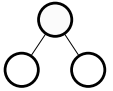
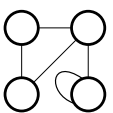
Underlying Action Structure	Effects/Preconditions types		
	Non-Factual Propositional	Factual Propositional	Factual Epistemic
Singleton 	NP-COMPLETE [Bolander et al., 2015]	PSPACE-HARD [Jensen, 2014]	PSPACE-HARD [Jensen, 2014]
Chain 	NP-COMPLETE [Bolander et al., 2015]	Open Question	Open Question
Tree 	PSPACE-COMPLETE [Bolander et al., 2015]	Open Question	Open Question
Graph 	EXSPACE [Bolander et al., 2015]	NON-ELEMENTARY [Yu et al., 2013]	UNDECIDABLE [Bolander and Andersen, 2011]

Table 1.3: Complexity of the *plan existence problem* [Bolander et al., 2015].

In Table 1.3 the complexity of the plan existence problem, depending on the action type and on the underlying action structure, is summarised. As expected, the complexity of the problem increases as we loosen the restrictions on the underlying structure. Unfortunately, we are interested in the subset of actions that are represented through graphs without restrictions and that have factual epistemic preconditions (the right-bottom cell of Table 1.3). While these results are on a general Kripke structure, *i.e.*, not constrained by any **S5** axiom, Bolander et al. [2015] show that even the plan existence problem on **S5_n**-states (more limited than **KD45_n**-states, in which we are mostly interested) is reducible to the *halting problem* that it is well known to be undecidable.

[...] the (unobserved) past, like the future, is indefinite and exists only as a spectrum of possibilities.

— Stephen Hawking
The Grand Design
[Hawking and Mlodinow, 2010]

2

Possibilities-Based MEP Action Language

Contents

2.1	Background	39
2.1.1	The Epistemic Action Language $m\mathcal{A}^*$	40
2.1.2	Possibilities	49
2.2	The Epistemic Action Language $m\mathcal{A}^p$	55
2.2.1	The Language Specification	56
2.2.2	The Language Properties	59
2.2.3	$m\mathcal{A}^*$ and $m\mathcal{A}^p$ Comparison	60

2.1 Background

While in Chapter 1 we presented a general introduction of the topics of this thesis, in this section we will explore in more detail the concepts that are required to describe the first contribution of our work, *i.e.*, the *multi-agent epistemic action language* $m\mathcal{A}^p$. We will start in Section 2.1.1 where we will present the MEP action language $m\mathcal{A}^*$ [Baral et al., 2015, Le et al., 2018, Baral et al., 2022], which has served as the foundation for our work. In Section 2.1.2 we will, then, introduce the theory of *non-well-founded* sets [Aczel, 1988] that is required to better understand the data structure that is used as a basis for $m\mathcal{A}^p$. Finally, again in Section 2.1.2, we will define formally the aforementioned data structure, referred to as *possibility*

in literature [Gerbrandy and Groeneveld, 1997, Gerbrandy, 1999], highlighting important properties that make it well-suited for representing e-states.

2.1.1 The Epistemic Action Language $m\mathcal{A}^*$

With the introduction of the classical planning problem, languages for representing actions and their effects were also proposed [Fikes and Nilsson, 1971]. These languages are referred to as *action languages* [Gelfond and Lifschitz, 1998].

Over the years, several action languages for single-agent scenarios (*e.g.*, STRIPS [Fikes and Nilsson, 1971], ADL [Pednault, 1994] and SAS+ [Bäckström, 1995]) have been developed providing the foundation for several successful approaches to automated planning. The effort of defining languages for classical planning domains culminated in the well-known *Planning Domain Description Language* (PDDL) [McDermott et al., 1998, Fox and Long, 2003] that standardized the notations and that is routinely adopted by planners. Nonetheless, as said by Baral et al. [2015]: “*in single-agent domains, reasoning about actions and change mainly involves reasoning about what is true in the world, what the agent knows about the world, how the agent can manipulate the world (using world-changing actions) to reach particular states, and how the agent (using sensing actions) can learn unknown aspects of the world.*”

On the other hand, multi-agent epistemic domains—the type of domain we are considering—need more careful consideration when it comes to actions effects. In particular, a MEP action language should be able to model how actions affect both the environment and the agents’ beliefs (about the environment or others’ beliefs). Similarly, the description of the states (be it an initial or a goal state) may involve the agents’ beliefs. Few studies directly address the challenges derived by domains in which information flows must be taken into consideration.

To the best of our knowledge, two works, *i.e.* Baral et al. [2015] and Muise et al. [2015], firstly tackled the problem of providing formal action languages for Multi-agent Epistemic Planning domains. In particular, in this section, we will

illustrate $m\mathcal{A}^*$ [Le et al., 2018, Baral et al., 2022]¹—the evolution of the language $m\mathcal{A}+$ provided by “*An Action Language for Multi-Agent Domains: Foundations*” by Baral et al. [2015]—as it is the foundation of our newly introduced language $m\mathcal{A}^o$.

We decided to develop a language starting from $m\mathcal{A}^*$, rather than PDKB-PDDL [Muise et al., 2015], because of the nature of the planning process employed by the planners related to these languages. In fact, we thought that $m\mathcal{A}^*$ to be more in line with our objective of defining a comprehensive epistemic environment that reasons on the full extent of \mathcal{L}_{AG}^C . While both planners can achieve these results, $m\mathcal{A}^*$ plans on a search space where each state is effectively a complete e-state, while PDKB-PDDL makes use of a conversion into classical planning. Intuitively, the latter *transforms* e-states properties into classical states to obtain a faster solving process but renouncing to reason on “full-fledged” epistemic models. We, therefore, preferred to define a system that, even if with a more resource-heavy procedure, is able to reason and update complete e-states representation, *e.g.*, Kripke structures.

Before formally introducing $m\mathcal{A}^*$ we need to provide some notations that are paramount to describe the language semantics. This introduction is supposed to provide the reader with enough information to understand, at an intuitive level, the characteristics of $m\mathcal{A}^*$ and, therefore, does not provide all the details of the language. For a complete analysis of the language, we address the interested reader to Baral et al. [2022] where $m\mathcal{A}^*$ is extensively analyzed.

The first idea that is necessary to introduce is the notion of *event model* (also called *update model*) [Baltag and Moss, 2004, Van Benthem et al., 2006]. In $m\mathcal{A}^*$, the event models are used to define how the execution of actions impacts an e-state, that is, they provide a formal way of defining how an action execution updates the epistemic states. Let us now define the concept of update models, along with the idea of *substitution* (necessary to define update models).

¹Let us note that we will use Baral et al. [2015] and Baral et al. [2022] as main references for $m\mathcal{A}^*$ as they define its syntax and semantics exhaustively. In fact, Le et al. [2018] only illustrate the additions to the language with respect to $m\mathcal{A}+$ redirecting the readers to Baral et al. [2015] for further information on the language.

Definition 2.1: $\mathcal{L}_{\mathcal{AG}}^{\mathcal{C}}$ -substitution [Baral et al., 2015]

Let $\mathcal{L}_{\mathcal{AG}}^{\mathcal{C}}$ be a language defined over a set \mathcal{AG} of n agents and a set \mathcal{F} of k fluents. An $\mathcal{L}_{\mathcal{AG}}^{\mathcal{C}}$ -substitution is a set $\{\mathbf{f}_1 \rightarrow \varphi_1, \dots, \mathbf{f}_k \rightarrow \varphi_k\}$, where each \mathbf{f}_i is a distinct proposition in \mathcal{F} and each $\varphi_i \in \mathcal{L}_{\mathcal{AG}}^{\mathcal{C}}$. We will implicitly assume that for each $\mathbf{f} \in \mathcal{F} \setminus \{\mathbf{f}_1, \dots, \mathbf{f}_k\}$, the substitution contains $\mathbf{f} \rightarrow \mathbf{f}$. $SUB_{(\mathcal{F}, \mathcal{AG})}$ denotes the set of all $\mathcal{L}_{\mathcal{AG}}^{\mathcal{C}}$ -substitutions.

Definition 2.2: Event Model [Baral et al., 2015]

Given a language $\mathcal{L}_{\mathcal{AG}}^{\mathcal{C}}$, defined over a set \mathcal{AG} of n agents and a set \mathcal{F} of k fluents, an *event model* Σ is a tuple $\langle \mathcal{E}, Q, pre, sub \rangle$ where:

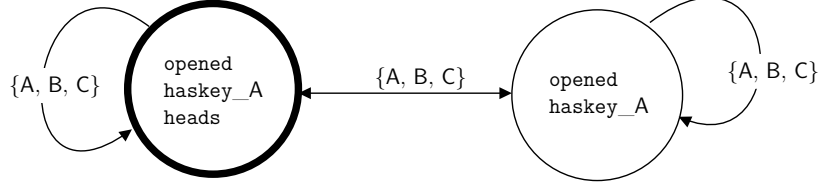
- \mathcal{E} : is a set, whose elements are called *events*;
- $Q: \mathcal{AG} \rightarrow 2^{\mathcal{E} \times \mathcal{E}}$ assigns an accessibility relation to each agent $i \in \mathcal{AG}$;
- $pre: \mathcal{E} \rightarrow \mathcal{L}_{\mathcal{AG}}^{\mathcal{C}}$ is a function mapping each event $e \in \mathcal{E}$ to a formula in $\mathcal{L}_{\mathcal{AG}}^{\mathcal{C}}$; and
- $sub: \mathcal{E} \rightarrow SUB_{(\mathcal{F}, \mathcal{AG})}$ is a function mapping each event $e \in \mathcal{E}$ to a substitution in $SUB_{(\mathcal{F}, \mathcal{AG})}$.

The idea behind event models is to provide a way to formally define an action that can correctly alter the underlying e-state representation (let us imagine a Kripke structure). Definition 2.3 illustrates how the information encoded in an update model (Figure 2.1b) is used to obtain the correct e-state update.

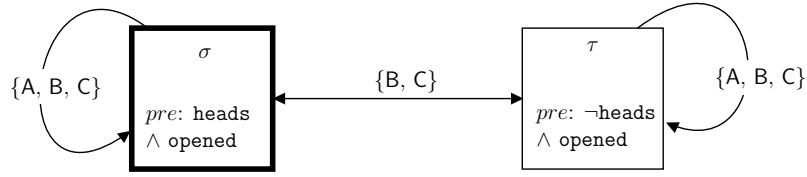
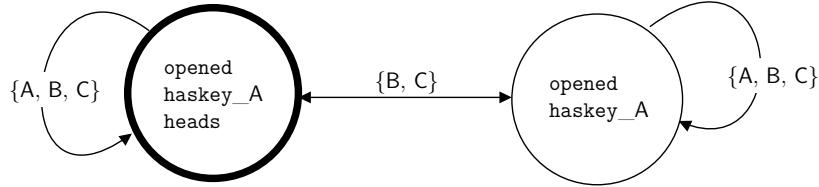
Definition 2.3: Update by Event Models [Baral et al., 2022]

Let (Σ, γ) be an *update template*, where $\Sigma = \langle \mathcal{E}, Q, pre, sub \rangle$ is an event model and $\gamma \in \mathcal{E}$, and $\text{lrt}(M, \mathbf{w})$ be an epistemic state. The execution of (Σ, γ) in (M, \mathbf{w}) results in an epistemic state $(M', \mathbf{w}) = (M, \mathbf{w}) \otimes (\Sigma, \gamma)$, where:

- $M'[W] = \{(t, e) \in M[W] \times \mathcal{E} \mid (M, t) \models pre(e)\}$;
- $M'[\mathbf{w}] = (M[\mathbf{w}], \gamma)$;
- $((t_1, e_1), (t_2, e_2)) \in M'[i]$ iff $(t_1, e_1), (t_2, e_2) \in M'[W]$, $(t_1, t_2) \in M[i]$ and $(e_1, e_2) \in Q$;
- For all $(\mathbf{w}, e) \in M'[W]$ and $\mathbf{f} \in \mathcal{F}$, $M'[\pi]((\mathbf{w}, e)) \models \mathbf{f}$ iff $\mathbf{f} \rightarrow \varphi \in sub(e)$ and $(M, \mathbf{w}) \models \varphi$.



(a) The e-state that represents the configuration where the box is opened

(b) The representation of the update template (Σ, σ) relative to $\text{peek}\langle A \rangle$. The substitutions are not indicated as they are equal to \emptyset .(c) The updated e-state after the execution of $\text{peek}\langle A \rangle$.**Figure 2.1:** The execution of an action instance through the application of Definition 2.3.

Let us note that, for simplicity, we assume that the event of interest, *i.e.*, γ , is exactly one and that the given e-state has one pointed world. These restrictions do not affect the definition of the language's properties needed in our introduction. Nonetheless, having a single-pointed world at each step means that the planning process implicitly discards the idea of executing conformant planning (where multiple unknown initial states should be kept into account). Since the language envisioned by Baral et al. is able to tackle also incomplete descriptions of the world, these assumptions are relaxed in their work [Baral et al., 2022].

Multiple events in a single update model correspond to multiple degrees of

observability; in particular Figure 2.1b represents the update model of the action instance `peek⟨A⟩`, introduced in Planning Domain 1.5. Here, only agent A becomes aware of the status of the coins, while the others learn that A knows the coin position without knowing it themselves—this corresponds to *partial observability*. We, therefore, have that the event σ corresponds to agent A learning the coin status, while event τ represents the other agents being aware of the peeking action. Figure 2.1 illustrates the result of applying the procedure described in Definition 2.3, with the update template in Figure 2.1b, to the Kripke structure that represents the state where the box is opened (Figure 2.1a) obtaining the correctly updated e-state (Figure 2.1c).

In what follows, we will introduce the syntax and the semantics of $m\mathcal{A}^*$ that will make use of more complex event models and observability relations. Once again, we will make use of the Coin in the Box domain. In particular, the example in Planning Domain 2.1 is a more complete version of the ones present in the previous chapter.

Each one of the actions presented in Planning Domain 2.1 falls into one of the three types distinguished by Baral et al. [2022]. In particular, these action types are:

- *World-altering* actions (also called *ontic*): used to modify certain properties (*i.e.*, fluents) of the world, *e.g.*, the actions `open` or `distract_X` of Planning Domain 2.1.
- *Sensing* action: used by an agent to refine her/his beliefs about the world, *e.g.*, the action `peek` of Planning Domain 2.1.
- *Announcement* action: used by an agent to affect the beliefs of other agents. *e.g.*, in Planning Domain 2.1 the action `announce`.

Planning Domain 2.1: Three Agents and the Coin in the Box

Three agents, **A**, **B**, and **C**, are in a room where in the middle there is a box. The box has a lock that can only be opened with a key. Inside the box, there is a coin that lies *heads* up. In the initial configuration of this domain we have that everybody knows that:

- none of the agents know whether the coin lies heads or tails up;
- the box is locked;
- only agent **A** has the key;
- if an agent is attentive (identified by `look_X` with $X \in \{A, B, C\}$) she/he is aware of the execution of the actions; and
- agents **A** and **C** are attentive while **B** is not.

Moreover, we have that each agent can execute one of the following actions:

- **open**: an agent, if she/he has the key, can open the box. This results in all the *attentive* agents believing that the box is open, while the others would not be aware of any change in the environment.
- **peek**: to learn whether the coin lies heads or tails up, an agent can peek into the box, but this requires the box to be open. This will result in the peeking agents believing the coin position while the other attentive agents are aware of this without knowing the coin position themselves.
- **announce**: this will result in all the listening (*i.e.*, attentive) agents to believe that the coin lies heads or tails up depending on the announced value. As before, for our configuration, this action is only executable by an agent who believes the coin position to be heads.
- **distract_X/signal_X**: these actions will make an attentive agent **X** no more attentive or vice-versa, respectively.

Finally, in the desired configuration, **A** would like to know whether the coin lies heads or tails up. She/He would also like to make agent **B** aware of this fact. However, **A** would like to keep this information secret from **C**.

A series of action instances—that is, a plan—to achieve the goal may be **(1) distract_C(A)**: to distract **C** from looking at the box; **(2) signal_B(A)**: to tell **B** to look at the box; **(3) open(A)**: to open the box; and **(4) peek(A)**: to make **A** peek into the box.

Given a domain D , an action instance $a \in D(\mathcal{AI})$, a fluent literal $f \in D(\mathcal{F})$, a fluent formula $\phi \in \mathcal{L}_{AG}^C$ and a belief formula $\varphi \in \mathcal{L}_{AG}^C$, where \mathcal{L}_{AG}^C is defined

Action type	Full observers	Partial Observers	Oblivious
World-altering	✓		✓
Sensing	✓	✓	✓
Announcement	✓	✓	✓

Table 2.1: Action types and observability relations Baral et al. [2015].

over $D(\mathcal{AG})$ and $D(\mathcal{F})$, we can “briefly” introduce the syntax adopted in $m\mathcal{A}^*$. Executability conditions are captured by statements of the form:

executable a if φ ;

for ontic actions we have:

a causes f if φ ;

sensing actions statements have the form expressed by:

a determines f;

finally, announcement actions are expressed as follows:

a announces ϕ .

In multi-agent domains, the execution of an action might change or not the beliefs of an agent. This is because, in such domains, each action instance associates an observability relation to each agent. For example, agent **C**—that becomes oblivious after being distracted by **A**—is not able to see the execution of the action $\text{open}\langle\mathbf{A}\rangle$. On the other hand, any agent who is watching a sensing or an announcement action can change her/his beliefs; *e.g.*, agent **B**, who is watching agent **A** sensing the status of the coin, will know that **A** knows the status of the coin without knowing it her/him-self. Table 2.1 summarizes the possible observability relations for each type of action. Partial observability for world-altering action is not admitted as, whenever an agent is aware of the execution of an ontic action, she/he must know its effects on the world as well. To indicate the set of agents that belong to the **Full**, **Partial**, and **Oblivious** observers we will use **F**, **P**, and **O**, respectively. The

idea of observability is captured in $m\mathcal{A}^*$ with specific statements. In particular, to state that an agent i is **Fully** observant, with respect to an action \mathbf{a} , it is used:

i observes \mathbf{a} if φ ;

while to identify an agent i as **Partially** observant, with respect to an action \mathbf{a} , it is used:

i aware_of \mathbf{a} if φ .

Notice that if we do not state otherwise, an agent will be considered oblivious. Finally, statements of the form

initially φ ;

and

goal φ

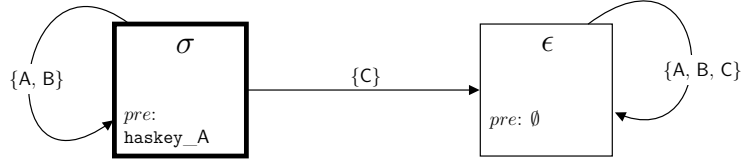
capture the initial and goal conditions, respectively.

The core of the language semantics is the transition function, and as already mentioned, it is defined using the concept of update models. In Figure 2.2a, Figure 2.2b, and Figure 2.2c we illustrate the update templates for ontic, sensing, and announcement actions, using **open** $\langle A \rangle$, **peek** $\langle A \rangle$, **announce** $\langle A \rangle$, respectively. The starting state is the one where **A** and **B** are attentive while **C** is not. That is, we can see **A** as representative for the fully observant, **B** as partially observant, and **C** as oblivious. For the sake of the presentation let us assume that **B** is partially observant also for the announcement action execution.

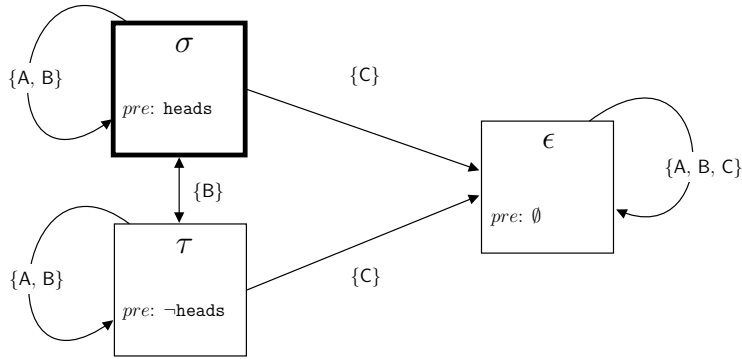
Finally, we can define the $m\mathcal{A}^*$ transition function Φ . Let (M, \mathbf{w}) be an e-state and let $\mathbf{a} \in D(\mathcal{AI})$. The result of executing \mathbf{a} on (M, \mathbf{w}) is the e-state, denoted by $\Phi(\mathbf{a}, (M, \mathbf{w}))$ defined as follow:

- If \mathbf{a} is not executable in (M, \mathbf{w}) then $\Phi(\mathbf{a}, (M, \mathbf{w})) = \emptyset$
- If \mathbf{a} is executable in (M, \mathbf{w}) and (Σ, σ) is the representation of the occurrence of \mathbf{a} on (M, \mathbf{w}) then $(M', w) = (M, \mathbf{w}) \otimes (\Sigma, \sigma)$.

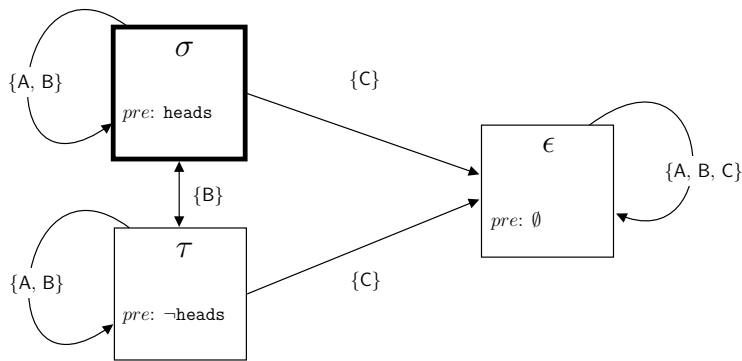
For more details and examples we address, once again, the reader to Baral et al. [2022].



(a) The update template (Σ, σ) of the ontic action instance $\text{open}\langle A \rangle$. The substitution in σ intuitively add the fluent haskey_A , and removes its negation. In ϵ the substitution is equal to \emptyset .



(b) The update template (Σ, σ) of the sensing action instance $\text{peek}\langle A \rangle$. The substitutions are equal to \emptyset .



(c) The update template (Σ, σ) of the announcement action instance $\text{announce}\langle A \rangle$. For the sake of the presentation we assume B to be partially observant. The substitutions are equal to \emptyset .

Figure 2.2: Examples of update templates for each action type described by Baral et al. [2022].

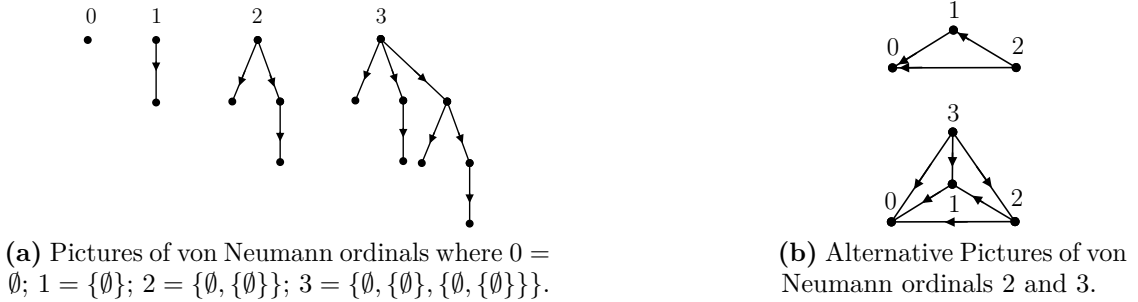


Figure 2.3: Well-founded sets represented through graphs [Aczel, 1988].

2.1.2 Possibilities

We are now ready to define the concept of *possibility* (originally introduced by Gerbrandy and Groeneveld [1997]). This section aims to provide the reader with enough information to understand what possibilities are, without providing all the details behind this topic. More on possibilities can be found in the works by Gerbrandy and Groeneveld [1997], Gerbrandy [1999], while we refer the reader to Aczel [1988] for a complete introduction on non-well-founded set theory and to Dovier [2015] for an introduction of non-well-founded sets and their equivalence in logic programming.

Non-well-founded Set Theory Fundamentals Let us start by giving some fundamental definitions of set theory. According to Aczel [1988], a *well-founded* and a *non-well-founded set* are defined as follows:

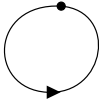
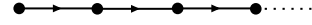
Definition 2.4: Well-founded Set

Let E be a set, E' one of its elements, E'' any element of E' , and so on. A descent is the sequence of steps from E to E' , E' to E'' , etc. ... A set is *well-founded* (or *ordinary*) when it only gives rise to finite descents.

Definition 2.5: Non-well-founded Set [Aczel, 1988]

A set is *non-well-founded* (or *extraordinary*) when among its descents there are some which are infinite.

All sets, in the sense of Definition 2.4, can be represented in the form of graphs, called *pictures*, as shown in Figure 2.3. The concept of *picture of a set* is introduced, alongside the definition of *decoration*, in Definition 2.6.

(a) Standard picture Ω .(b) Unfolding of the picture of Ω .**Figure 2.4:** Representation of the non-well-founded set $\Omega = \{\Omega\}$ [Aczel, 1988].**Definition 2.6: Decoration and Picture**

- A *decoration* of a graph $\mathcal{G} = (V, E)$ is a function δ that assigns to each node $n \in V$ a set δ_n in such a way that the elements of δ_n are exactly the sets assigned to successors of n , *i.e.*, $\delta_n = \{\delta_{n'} \mid (n, n') \in E\}$. Therefore, the edges denote the membership relations.
- If δ is a decoration of a pointed graph (\mathcal{G}, n) , then (\mathcal{G}, n) is a *picture* of the set δ_n .

These concepts are essential to investigate the differences between well-founded and non-well-founded set theories. We know that in well-founded set theory, it holds Mostowski’s lemma: “*each well-founded graph² is a picture of exactly one set*” [Mostowski, 1949]. On the other hand, when the **Foundation Axiom**, expressed by Gerbrandy [1999] as “*Only well-founded graphs have decorations*”, is substituted with the **Anti-Foundation Axiom (AFA)**, expressed by Aczel [1988] as “*Every graph has a unique decoration*”, the following consequences become true:

- every graph is a picture of exactly one set (**AFA** as is formulated by Gerbrandy [1999]);
- non-well-founded sets exist given that a non-well-founded pointed graph has to be a picture of a non-well-founded set.

Aczel [1988], Gerbrandy [1999] point out how non-well-founded sets can also be expressed through systems of equations. This concept will help us to formalize the notion of state in our action language $m\mathcal{A}^p$. A quick example of this representation can be derived by the set $\Omega = \{\Omega\}$ (Figure 2.4). We can informally define this set

²A well-founded graph is a graph that doesn’t contain an infinite path $n \rightarrow n' \rightarrow n'' \rightarrow \dots$ of successors.

as the (singleton) system of equations $x = \{x\}$. Systems of equations and their solutions are described more formally in Definition 2.7.

Definition 2.7: System of Equations [Gerbrandy, 1999]

For each class of atoms, *i.e.*, objects that are not sets and have no further set-theoretic structure, \mathcal{X} a *system of equations* in \mathcal{X} is a class τ of equations $x = X$, where $x \in \mathcal{X}$ and $X \subseteq \mathcal{X}$, such that τ contains exactly one equation $x = X$ for each $x \in \mathcal{X}$. A solution to a system of equations τ is a function δ that assigns to each $x \in \tau(\mathcal{X})^a$ a set δ_x such that $\delta_x = \{\delta_y \mid y \in X\}$, where $x = X$ is an equation of τ . If δ is the solution to a system of equations τ , then the set $\{\delta_x \mid x \in \tau(\mathcal{X})\}$ is called the solution set of that system.

^a $\tau(\mathcal{X})$ denotes the class of atoms \mathcal{X} in which τ is described.

Since both graphs and systems of equations are representations for non-well-founded sets, it is natural to investigate their relationships. In particular, it is interesting to point out how from a graph $\mathcal{G} = (V, E)$ it is possible to construct a system of equations τ and vice versa. The nodes in \mathcal{G} , in fact, can be the set of atoms $\tau(\mathcal{X})$ and, for each node $v \in V$, an equation is represented by $v = \{v' \mid (v, v') \in E\}$. Since each graph has a unique decoration, each system of equations has a unique solution. Nonetheless, different graphs can represent the same set, and the notion that can help to identify this equivalence is known as *bisimulation*.

Bisimulation As mentioned before the idea of *bisimulation* can be exploited to characterize graphs, and therefore Kripke models, with “the same behavior”. In particular, it can be proved that there is a unique minimum graph bisimilar to a given one, and it can be found by computing the *maximum bisimulation*. Bisimilar labeled graphs (or Kripke structures) have therefore a unique solution as well since we collapse their representations into the minimal one. While this topic is of the utmost importance in modal logic, and has been studied and used in several fields, it is not the aim of this thesis to study it in depth. Let us, therefore, only provide some basic notions referring the interested readers to much more complete works such as Gerbrandy [1999], Bolander et al. [2015], Dovier [2015]. In particular, Definition 2.8 formally introduces the concepts of bisimulation and Figure 2.5

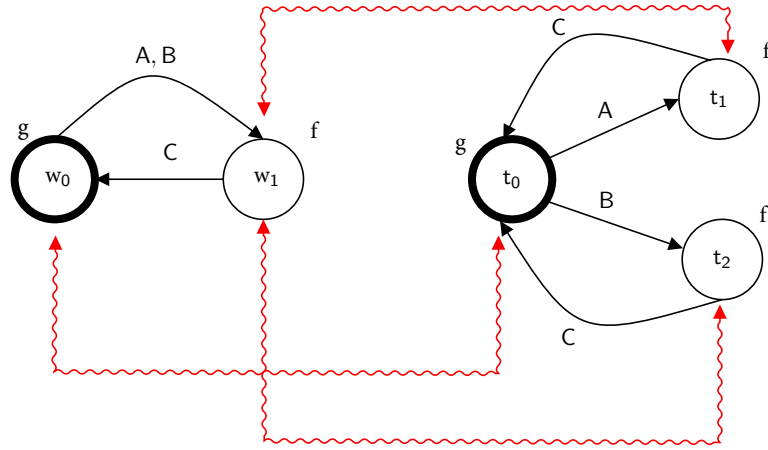


Figure 2.5: In Figure are represented two different pointed Kripke structures (M_1, w_0) (left) and (M_2, t_0) (right). It is easy to see that these two Kripke structures are structurally different but bisimilar since there exists a relation $\{(w_0, t_0), (w_1, t_1), (w_1, t_2)\}$ that is a bisimulation [Riouak, 2019].

presents a graphical example of such concept. Finally, Corollary 2.1 states how the concept of entailment and bisimulation are intertwined in Kripke structures.

Definition 2.8: Bisimulation [Bolander et al., 2015]

Given two pointed Kripke structures (M_1, w_0) and (M_2, t_0) let $W = M_1[W]$ and $T = M_2[W]$, and let \mathcal{F} be a set of propositional variables. The relation $\mathcal{R} \subseteq W \times T$ is a *bisimulation* if and only if for all $w \in W$ and $t \in T$, if $(w, t) \in \mathcal{R}$ then:

- **Atom:** $w \models f \iff t \models f$, for all $f \in \mathcal{F}$;
- **Forth:** For all w' such that $(w, w') \in M_1[\mathcal{B}]$ there exists a t' such that $(t, t') \in M_2[\mathcal{B}]$ and $(w', t') \in \mathcal{R}$;
- **Back:** For all t' such that $(t, t') \in M_2[\mathcal{B}]$ there exists a w' such that $(w, w') \in M_1[\mathcal{B}]$ and $(t', w') \in \mathcal{R}$;

Two pointed Kripke structures (M_1, w_0) and (M_2, t_0) are *bisimilar*, indicated with $(M_1, w_0) \simeq (M_2, t_0)$, if and only if there exists a bisimulation between M_1 and M_2 such that $(s_0, t_0) \in \mathcal{R}$.

Corollary 2.1: Truth in Bisimilar Kripke Structures

Given two pointed Kripke structures (M_1, w_0) and (M_2, t_0) and a language \mathcal{L}_{AG}^C , we have that:

$$(M_1, w_0) \simeq (M_2, t_0) \iff (\forall \varphi \in \mathcal{L}_{AG}^C : (M_1, w_0) \models \varphi \iff (M_2, t_0) \models \varphi).$$

Possibilities Let us now introduce the notion of possibility, following Gerbrandy and Groeneveld [1997]:

Definition 2.9: Possibilities [Gerbrandy and Groeneveld, 1997]

Let \mathcal{AG} be a set of agents and \mathcal{F} a set of propositional variables:

- A *possibility* \mathbf{u} is a function that assigns to each propositional variable $\mathbf{f} \in \mathcal{F}$ a truth value $\mathbf{u}(\mathbf{f}) \in \{0, 1\}$ and to each agent $i \in \mathcal{AG}$ an information state $\mathbf{u}(i) = \sigma$.
- An *information state* σ is a set of possibilities.

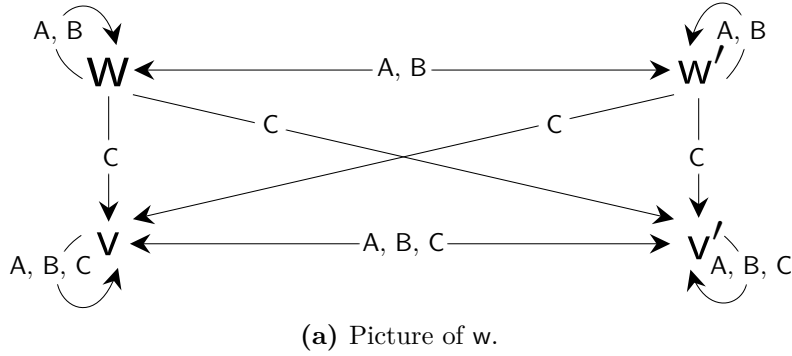
In Section 2.2 we will use this concept to describe an epistemic state. The intuition behind this is that a possibility \mathbf{u} is a “possible” interpretation of the world and of the agents’ beliefs. That is, $\mathbf{u}(\mathbf{f})$ specifies the truth value of the fluent \mathbf{f} in \mathbf{u} and $\mathbf{u}(a)$ is the set of all the interpretations that agent a considers possible in \mathbf{u} .

Moreover, a possibility can be represented as a decoration (Definition 2.6) of a labeled graph and, therefore, as a unique solution to a system of equations for possibilities (Definition 2.10), as shown in Figure 2.6. A possibility represents the solution to the minimal system of equations in which all bisimilar systems of equations are collapsed; namely, the possibilities that represent decorations of bisimilar labeled graphs are bisimilar and can be represented by the minimal one. This shows that the class of bisimilar labeled graphs and, therefore, of bisimilar Kripke structures, used by $m\mathcal{A}^*$ as e-states, can be represented by a single possibility.

Definition 2.10: System of Equations for Possibilities [Gerbrandy, 1999]

Given a set of agents \mathcal{AG} and a set of propositional variables \mathcal{F} , a *system of equations for possibilities* in a class of possibilities \mathcal{X} is a set of equations such that for each $\mathbf{x} \in \mathcal{X}$ there exists exactly one equation of the form $\mathbf{x}(\mathbf{f}) = b$, where $b \in \{0, 1\}$, for each $\mathbf{f} \in \mathcal{F}$, and of the form $\mathbf{x}(i) = \mathbf{X}$, where $\mathbf{X} \subseteq \mathcal{X}$, for each $i \in \mathcal{AG}$.

A solution to a system of equations for possibilities is a function δ that assigns to each atom \mathbf{x} a possibility $\delta_{\mathbf{x}}$ in such a way that if $\mathbf{x}(\mathbf{f}) = i$ is an equation then $\delta_{\mathbf{x}(\mathbf{f})} = i$, and if $\mathbf{x}(i) = \sigma$ is an equation, then $\delta_{\mathbf{x}(i)} = \{\delta_{\mathbf{y}} \mid \mathbf{y} \in \sigma\}$.



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

(b) System of equations of w .

Figure 2.6: Representation of a generic possibility w . The possibility is expanded for clarity.

Finally, in Proposition 2.1, we show some interesting relations between labeled graphs and possibilities, while in Proposition 2.2 we summarize important properties that capture the relations between Kripke structures and possibilities.

Proposition 2.1: Labeled Graphs and Possibilities
[Gerbrandy and Groeneveld, 1997]

The relations between labeled graphs and possibilities are summarized as follows:

- *each possibility can be pictured by a labeled graph;*
- *each labeled graph has a unique decoration;*
- *two labeled graphs have the same decoration if and only if are bisimilar.*

Proposition 2.2: Kripke Structures and Possibilities

Given a Kripke structure (M, \mathbf{w}) , a possibility \mathbf{u} , and a language \mathcal{L}_{AG}^C , we have that:

- *each pointed Kripke structure has exactly one set as its solution;*
- *two models are bisimilar if and only if they are the picture of the same set; and*
- *If (M, \mathbf{w}) is a picture of \mathbf{u} , then for each $\varphi \in \mathcal{L}_{AG}^C$ it holds $(M, \mathbf{w}) \models \varphi \iff \mathbf{u} \models \varphi$.*

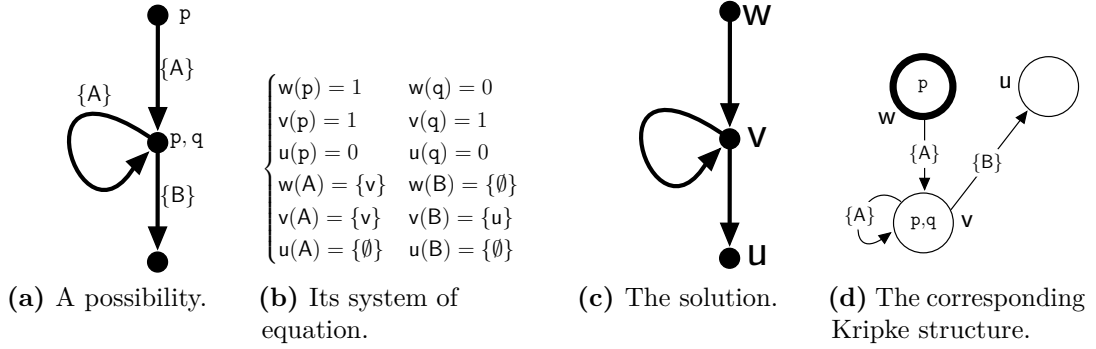


Figure 2.7: An e-state represented through a possibility (a) and then converted to a Kripke structure (d).

2.2 The Epistemic Action Language $m\mathcal{A}^\rho$

The aforementioned idea of possibilities is central in $m\mathcal{A}^\rho$. This language, instead of using Kripke structures, exploits possibilities as e-states (Figure 2.7). That is, $m\mathcal{A}^\rho$, while keeping the same syntax of $m\mathcal{A}^*$, changes the way of representing an epistemic state. The modification of the underlying structure implies also a different formalization of the transition function. This allowed us to define a new planning environment that outperforms the state-of-the-art comprehensive epistemic planner presented by [Le et al., 2018] by orders of magnitude in the experiments.

We will now briefly explain how a possibility can be used to represent an e-state. The main idea is to identify with each possibility u both an interpretation of the world and of each agent’s beliefs. That is, the component $u(\mathbf{f})$ assigns a truth value to the fluent \mathbf{f} in u while $u(i)$ represents the (non-well-founded) set of possibilities that are considered by agent i .

The choice of possibilities over Kripke structures as e-state representation provides several advantages. One of these is, as said by Gerbrandy and Groeneveld [1997], that: “a possibility represents the solution to the minimal system of equations in which all bisimilar Kripke structures are collapsed”. Intuitively, this means that a class of bisimilar Kripke structures, that in $m\mathcal{A}^*$ represents different e-states, is easily represented by a single possibility and therefore, by a single e-state in $m\mathcal{A}^\rho$. That is, thanks to possibilities and the newly introduced transition function

it has been possible to maintain e-states with smaller size, with respect to the planner EFP 1.0 presented by Le et al. [2018], during the solving process. From a more concrete point of view, implementing $m\mathcal{A}^p$ allowed us to work on e-states of reduced dimension³ without having to rely on minimization techniques, such as the algorithms presented by Paige and Tarjan [1987], Dovier et al. [2004], during the solving process. Another advantage of using possibilities derives from their non-well-founded aspect. Since a possibility is a non-well-founded graph, whose nodes are themselves possibilities, the solving process can store each calculated possibility; and, whenever needed, it can retrieve the stored possibilities to reuse them as “nodes” inside a new e-state. To summarize, although possibilities and Kripke structures are tightly connected (Figure 2.7), the advantages of using $m\mathcal{A}^p$ are: (i) the reduced size of the e-states that does not depend on external procedures; and (ii) the fact that possibilities can be stored and easily reused thanks to their non-well-founded nature. In this sense, we can see possibilities as a more compact representation, with respect to Kripke structures, that allows us to save computational resources.

2.2.1 The Language Specification

As the first main contribution, we present the language $m\mathcal{A}^p$ ⁴. $m\mathcal{A}^p$ borrows the syntax from $m\mathcal{A}^*$ but changes the underlying e-state representation from Kripke structures to possibilities. After rapidly introducing the concept of entailment we will describe an improved transition function for $m\mathcal{A}^p$ along with some important properties.

Let us start with the concept of entailment for possibilities. Definition 2.11 combines the concept of Gerbrandy [1999] with the action language $m\mathcal{A}^*$.

³With respect to the e-states generated following $m\mathcal{A}^*$.

⁴The original version of $m\mathcal{A}^p$ was presented by Fabiano et al. [2019]. Instead, here we will introduce a newer version that maintains the same core while optimizing some details.

Definition 2.11: Entailment in Possibilities

Given, a fluent \mathbf{f} , the belief formulae $\varphi, \varphi_1, \varphi_2$, an agent i , a group of agents α , and a possibility \mathbf{u} :

- (i) $\mathbf{u} \models \mathbf{f}$ if $\mathbf{u}(\mathbf{f}) = 1$;
- (ii) $\mathbf{u} \models \varphi$ if φ is a fluent formula and $\mathbf{u} \models \varphi$ following the standard semantics for \neg and \wedge ;
- (iii) $\mathbf{u} \models \mathbf{B}_i(\varphi)$ if for each $\mathbf{v} \in \mathbf{u}(i)$, $\mathbf{v} \models \varphi$;
- (iv) $\mathbf{u} \models \neg\varphi$ if $\mathbf{u} \not\models \varphi$;
- (v) $\mathbf{u} \models \varphi_1 \wedge \varphi_2$ if $\mathbf{u} \models \varphi_1$ and $\mathbf{u} \models \varphi_2$;
- (vi) $\mathbf{u} \models \mathbf{E}_\alpha\varphi$ if $\mathbf{u} \models \mathbf{B}_i(\varphi)$ for all $i \in \alpha$;
- (vii) $\mathbf{u} \models \mathbf{C}_\alpha(\varphi)$ if $\mathbf{u} \models \mathbf{E}_\alpha^k\varphi$ for every $k \geq 0$, where $\mathbf{E}_\alpha^0\varphi = \varphi$ and $\mathbf{E}_\alpha^{k+1}\varphi = \mathbf{E}_\alpha(\mathbf{E}_\alpha^k\varphi)$.

We are now ready to introduce the transition function. This new transition function is, in our opinion, more compact and therefore, more approachable than the one introduced by Le et al. [2018]. Moreover, the ‘‘simplicity’’ of the e-states update formalization is reflected in a much cleaner and faster implementation, as we will see in Chapter 5. Let a domain D , its set of action instances $D(\mathcal{AI})$, and the set \mathcal{S} of all the possibilities reachable from $D(\varphi_{ini})$ with a finite sequence of action instances be given. Moreover let us identify the observability groups \mathbf{F} , \mathbf{P} , and \mathbf{O} with respect to an action instance \mathbf{a} with $\mathbf{F}_\mathbf{a}$, $\mathbf{P}_\mathbf{a}$, and $\mathbf{O}_\mathbf{a}$, respectively. The transition function $\Phi : D(\mathcal{AI}) \times \mathcal{S} \rightarrow \mathcal{S} \cup \{\emptyset\}$ for $m\mathcal{A}^\rho$ relative to D is formalized following Definition 2.12.

Definition 2.12: $m\mathcal{A}^\rho$ transition function

Allow us to use the compact notation $\mathbf{u}(\mathcal{F}) = \{\mathbf{f} \mid \mathbf{f} \in D(\mathcal{F}) \wedge \mathbf{u} \models \mathbf{f}\} \cup \{\neg\mathbf{f} \mid \mathbf{f} \in D(\mathcal{F}) \wedge \mathbf{u} \not\models \mathbf{f}\}$ for the sake of readability. Let an action instance $\mathbf{a} \in D(\mathcal{AI})$, a possibility $\mathbf{u} \in \mathcal{S}$ and an agent $i \in D(\mathcal{AG})$ be given. If \mathbf{a} is not executable in \mathbf{u} , then $\Phi(\mathbf{a}, \mathbf{u}) = \emptyset$ otherwise $\Phi(\mathbf{a}, \mathbf{u}) = \mathbf{u}'$, where:

- Let us consider the case of an *ontic* action instance \mathbf{a} . We then define \mathbf{u}'

such that:

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

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

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

$$\mathbf{u}'(i) = \begin{cases} \mathbf{u}(i) & \text{if } i \in \mathbf{O}_a \\ \bigcup_{w \in \mathbf{u}(i)} \Phi(\mathbf{a}, w) & \text{if } i \in \mathbf{F}_a \end{cases}$$

- if \mathbf{a} is a *sensing* action instance, used to determine the fluent \mathbf{f} . We then define \mathbf{u}' such that:

$$e(\mathbf{a}, \mathbf{u}) = \{\mathbf{f} \mid (\mathbf{a} \text{ determines } \mathbf{f}) \in D \wedge \mathbf{u} \models \mathbf{f}\} \\ \cup \{\neg\mathbf{f} \mid (\mathbf{a} \text{ determines } \mathbf{f}) \in D \wedge \mathbf{u} \not\models \mathbf{f}\}$$

$$\mathbf{u}'(\mathcal{F}) = \mathbf{u}(\mathcal{F})$$

$$\mathbf{u}'(i) = \begin{cases} \mathbf{u}(i) & \text{if } i \in \mathbf{O}_a \\ \bigcup_{w \in \mathbf{u}(i)} \Phi(\mathbf{a}, w) & \text{if } i \in \mathbf{P}_a \\ \bigcup_{w \in \mathbf{u}(i): e(\mathbf{a}, w) = e(\mathbf{a}, \mathbf{u})} \Phi(\mathbf{a}, w) & \text{if } i \in \mathbf{F}_a \end{cases}$$

- if \mathbf{a} is an *announcement* action instance of the fluent formula ϕ . We then define \mathbf{u}' such that:

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

$$\mathbf{u}'(\mathcal{F}) = \mathbf{u}(\mathcal{F})$$

$$\mathbf{u}'(i) = \begin{cases} \mathbf{u}(i) & \text{if } i \in \mathbf{O}_a \\ \bigcup_{w \in \mathbf{u}(i)} \Phi(\mathbf{a}, w) & \text{if } i \in \mathbf{P}_a \\ \bigcup_{w \in \mathbf{u}(i): e(\mathbf{a}, w) = e(\mathbf{a}, \mathbf{u})} \Phi(\mathbf{a}, w) & \text{if } i \in \mathbf{F}_a \end{cases}$$

2.2.2 The Language Properties

The newly introduced transition function allowed us to reason about fundamental properties that, as said by Baral et al. [2015], each multi-agent epistemic action language should respect. In particular, each epistemic reasoner should ensure that:

- if an agent is fully aware of the execution of an action instance then her/his beliefs will be updated with the effects of such action execution;
- an agent who is only partially aware of the action occurrence will believe that the agents who are fully aware of the action occurrence are certain about the effects of the actions; and
- an agent who is oblivious of the action occurrence will also be ignorant about its effects.

Propositions 2.3 to 2.5 capture the concept of beliefs update and ensure that, when satisfied, the action language can be soundly used for multi-agent epistemic reasoning. For the sake of readability, their complete proofs are reported in Appendix A.2. In the following, we will use \mathbf{p}' instead of $\Phi(\mathbf{a}, \mathbf{p})$ when possible to avoid unnecessary clutter.

Proposition 2.3: Ontic Action Properties

Assume that \mathbf{a} is an ontic action instance executable in \mathbf{u} s.t. \mathbf{a} **causes** ℓ if ψ belongs to D . In $m\mathcal{A}^p$ it holds that:

- (1) for every agent $\mathbf{x} \in \mathbf{F}_{\mathbf{a}}$, if $\mathbf{u} \models \mathbf{B}_{\mathbf{x}}(\psi)$ then $\mathbf{u}' \models \mathbf{B}_{\mathbf{x}}(\ell)$;
- (2) for every agent $\mathbf{y} \in \mathbf{O}_{\mathbf{a}}$ and a belief formula φ , $\mathbf{u}' \models \mathbf{B}_{\mathbf{y}}(\varphi)$ iff $\mathbf{u} \models \mathbf{B}_{\mathbf{y}}(\varphi)$;
and
- (3) for every pair of agents $\mathbf{x} \in \mathbf{F}_{\mathbf{a}}$ and $\mathbf{y} \in \mathbf{O}_{\mathbf{a}}$ and a belief formula φ , if $\mathbf{u} \models \mathbf{B}_{\mathbf{x}}(\mathbf{B}_{\mathbf{y}}(\varphi))$ then $\mathbf{u}' \models \mathbf{B}_{\mathbf{x}}(\mathbf{B}_{\mathbf{y}}(\varphi))$.

Proposition 2.4: Sensing Action Properties

Assume that \mathbf{a} is a sensing action instance and D contains the statement \mathbf{a} *determines* \mathbf{f} . In $m\mathcal{A}^\rho$ it holds that:

- (1) if $u \models \mathbf{f}$ then $u' \models \mathbf{C}_{\mathbf{F}_a} \mathbf{f}$;
- (2) if $u \models \neg \mathbf{f}$ then $u' \models \mathbf{C}_{\mathbf{F}_a} \neg \mathbf{f}$;
- (3) $u' \models \mathbf{C}_{\mathbf{P}_a} (\mathbf{C}_{\mathbf{F}_a} \mathbf{f} \vee \mathbf{C}_{\mathbf{F}_a} \neg \mathbf{f})$;
- (4) $u' \models \mathbf{C}_{\mathbf{F}_a} (\mathbf{C}_{\mathbf{P}_a} (\mathbf{C}_{\mathbf{F}_a} \mathbf{f} \vee \mathbf{C}_{\mathbf{F}_a} \neg \mathbf{f}))$;
- (5) for every agent $\mathbf{y} \in \mathbf{O}_a$ and a belief formula φ , $u' \models \mathbf{B}_y(\varphi)$ iff $u \models \mathbf{B}_y(\varphi)$;
and
- (6) for every pair of agents $\mathbf{x} \in \mathbf{F}_a$ and $\mathbf{y} \in \mathbf{O}_a$ and a belief formula φ , if
 $u \models \mathbf{B}_x(\mathbf{B}_y(\varphi))$ then $u' \models \mathbf{B}_x(\mathbf{B}_y(\varphi))$.

Proposition 2.5: Announcement Action Properties

Assume that \mathbf{a} is a announcement action instance and D contains the statement \mathbf{a} *announces* φ . If $u \models \varphi$ in $m\mathcal{A}^\rho$ it holds that:

- (1) $u' \models \mathbf{C}_{\mathbf{F}_a} \varphi$;
- (2) $u' \models \mathbf{C}_{\mathbf{P}_a} (\mathbf{C}_{\mathbf{F}_a} \varphi \vee \mathbf{C}_{\mathbf{F}_a} \neg \varphi)$;
- (3) $u' \models \mathbf{C}_{\mathbf{F}_a} (\mathbf{C}_{\mathbf{P}_a} (\mathbf{C}_{\mathbf{F}_a} \varphi \vee \mathbf{C}_{\mathbf{F}_a} \neg \varphi))$;
- (4) for every agent $\mathbf{y} \in \mathbf{O}_a$ and a belief formula φ , $u' \models \mathbf{B}_y(\varphi)$ iff $u \models \mathbf{B}_y(\varphi)$;
and
- (5) for every pair of agents $\mathbf{x} \in \mathbf{F}_a$ and $\mathbf{y} \in \mathbf{O}_a$ and a belief formula φ , if
 $u \models \mathbf{B}_x(\mathbf{B}_y(\varphi))$ then $u' \models \mathbf{B}_x(\mathbf{B}_y(\varphi))$.

Baral et al. [2015] show how the above-listed properties capture the concept of update in an epistemic environment. Therefore, we consider the epistemic action languages, that respect all of the aforementioned properties, to be correct with respect to the knowledge/belief update. That is the case with both $m\mathcal{A}^*$ and $m\mathcal{A}^\rho$.

2.2.3 $m\mathcal{A}^*$ and $m\mathcal{A}^\rho$ Comparison

Finally, for a clearer understanding on differences between the two languages, let us show (i) the execution, on both $m\mathcal{A}^*$ and $m\mathcal{A}^\rho$, of the action instances sequence Δ ;

	distract_C⟨A⟩	open⟨A⟩	peek⟨A⟩
F_D	A, B, C	A, B	A
P_D	-	-	B
O_D	-	C	C

Table 2.2: Observability relations of the actions instances in Δ .

and (ii) a direct comparison of the number of worlds and edges created by $m\mathcal{A}^*$ and $m\mathcal{A}^p$ when executing a sequence of action instances.

Plan Execution $\Delta = \text{distract_C}\langle A \rangle, \text{open}\langle A \rangle, \text{peek}\langle A \rangle$ is the sequence that leads to the desired goal in Planning Domain 2.1 if we assume that B is already looking at the box. With this, we want to give a graphical explanation of both the transition functions and state-space defined by the two languages (Figures 2.8 to 2.11). Each state in $m\mathcal{A}^*$ will be represented by a Kripke structure while in $m\mathcal{A}^p$ will be a possibility (expanded to its respective system of equations for clarity). The observability relations of each action instance in Δ are expressed in Table 2.2.

Assuming that $\alpha = \{A, B, C\}$, then the initial state, based on a small variation of Planning Domain 2.1 where we assume B to be already attentive, is defined by the conditions:

- **initially** $C_\alpha(\text{haskey_A}) \wedge C_\alpha(\neg \text{haskey_B}) \wedge C_\alpha(\neg \text{haskey_C})$
- **initially** $C_\alpha(\neg \text{opened})$
- **initially** $C_\alpha(\neg \mathbf{B}_i(\text{heads}) \wedge \neg \mathbf{B}_i(\neg \text{heads}))$ for $i \in \alpha$
- **initially** $C_\alpha(\text{look_i})$ for $i \in \alpha$
- **initially** heads

Finally, the goal of Planning Domain 2.1 is expressed with the following formulae:

$$\begin{aligned}
& \mathbf{B}_A(\text{heads}) \wedge \mathbf{B}_A(\mathbf{B}_B((\mathbf{B}_A(\text{heads}) \vee \mathbf{B}_A(\neg \text{heads})))) \\
& \mathbf{B}_B(\mathbf{B}_A(\text{heads}) \vee \mathbf{B}_A(\neg \text{heads})) \wedge (\neg \mathbf{B}_B(\text{heads} \wedge \neg \mathbf{B}_B(\neg \text{heads}))) \\
& \mathbf{B}_C([\bigwedge_{i \in \{A, B, C\}} (\neg \mathbf{B}_i(\text{heads}) \wedge \neg \mathbf{B}_i(\neg \text{heads}))])
\end{aligned}$$

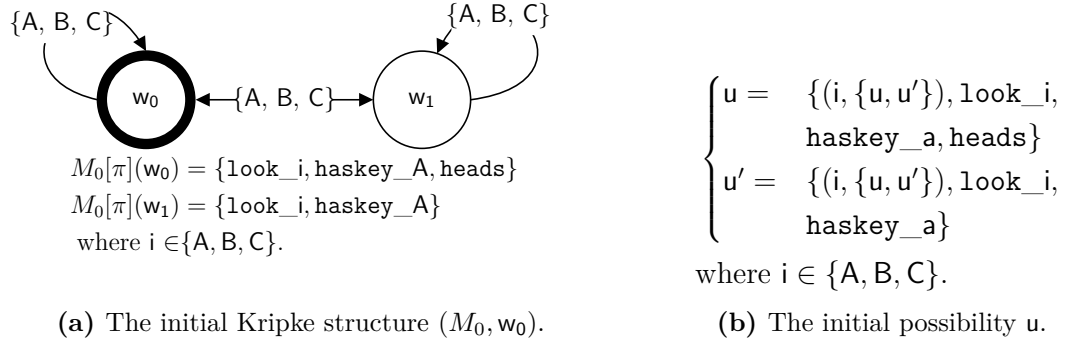
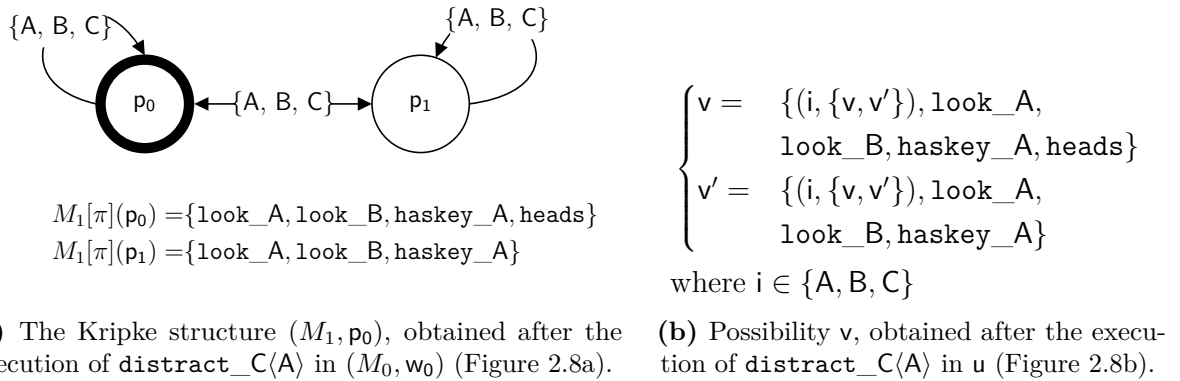
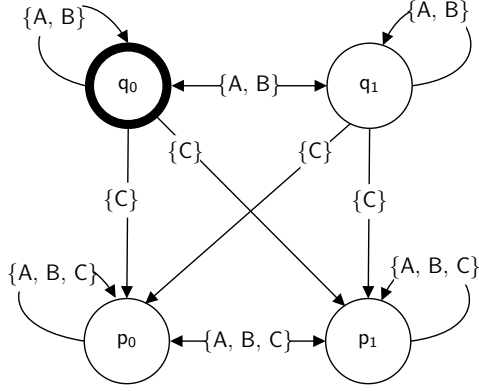


Figure 2.8: The initial state.

Figure 2.9: Execution of $\text{distract_C}\langle A \rangle$.



$M_2[\pi](q_0) = \{\text{look}_i, \text{haskey_A}, \text{opened}, \text{heads}\}$
 $M_2[\pi](q_1) = \{\text{look}_i, \text{haskey_A}, \text{opened}\}$
 where $i \in \{A, B\}$.

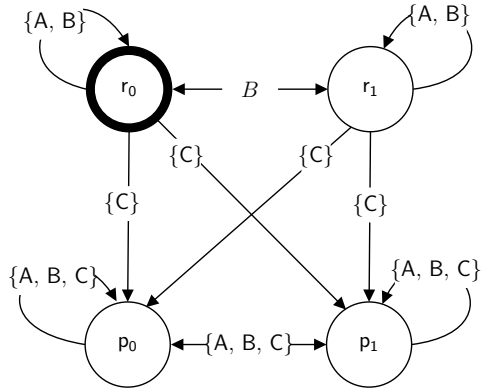
$$\begin{cases} w = \{(i, \{w, w'\}), (C, \{v, v'\}), \text{look}_i, \\ \text{haskey_a}, \text{opened}, \text{heads}\} \\ w' = \{(i, \{w, w'\}), (C, \{v, v'\}), \text{look}_i, \\ \text{haskey_a}, \text{opened}\} \end{cases}$$

where $i \in \{A, B\}$ and v, v' are the possibilities of Figure 2.9b.

(a) The Kripke structure (M_2, q_0) , obtained after the execution of $\text{open}\langle A \rangle$ in (M_1, p_0) (Figure 2.9a).

(b) Possibility w , obtained after the execution of $\text{open}\langle A \rangle$ in v (Figure 2.9b).

Figure 2.10: Execution of $\text{open}\langle A \rangle$.



$M_3[\pi](r_0) = \{\text{look}_i, \text{haskey_A}, \text{opened}, \text{heads}\}$
 $M_3[\pi](r_1) = \{\text{look}_i, \text{haskey_A}, \text{opened}\}$
 where $i \in \{A, B\}$.

$$\begin{cases} z = \{(A, \{z\}), (B, \{z, z'\}), (C, \{v, v'\}), \\ \text{look}_i, \text{haskey_a}, \text{opened}, \text{heads}\} \\ z' = \{(A, \{z'\}), (B, \{z, z'\}), (C, \{v, v'\}), \\ \text{look}_i, \text{haskey_a}, \text{opened}\} \end{cases}$$

where $i \in \{A, B\}$ and the possibilities v, v' are represented in Figure 2.9b.

(a) The Kripke structure (M_3, q_0) , obtained after the execution of $\text{peek}\langle A \rangle$ in (M_2, q_0) .

(b) Possibility z , obtained after the execution of $\text{peek}\langle A \rangle$ in w (Figure 2.10b).

Figure 2.11: Execution of $\text{peek}\langle A \rangle$.

e-States Size Comparison To conclude, let us summarize, in Figure 2.12, the difference in size of the e-states of created by $m\mathcal{A}^\rho$ and $m\mathcal{A}^*$. This figure gives an intuitive idea of how $m\mathcal{A}^\rho$ helps in reducing the e-state size and its relative overhead. We note, in fact, that the size of the e-states, generated after ten consecutive action instances execution, varies immensely between the one generated by $m\mathcal{A}^\rho$, that is comprised of 59 worlds and 291 edges, and the one produced by $m\mathcal{A}^*$ that has 1461 worlds and 8037 edges. Let us stress that with a smaller e-state size is associated a faster solving process as we will show in the experimental evaluation, later in this thesis (Chapter 5).

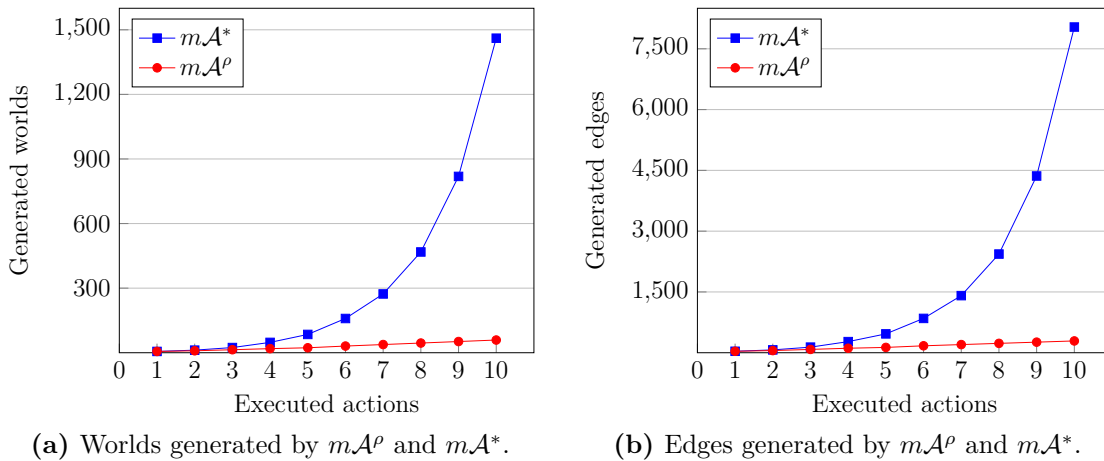


Figure 2.12: Comparison between the number of worlds and edges generated by $m\mathcal{A}^\rho$ and $m\mathcal{A}^*$ on the Coin in the Box domain with a sequence of ten action instances.

Fidarsi è bene, non fidarsi è meglio.

To trust is good, not to trust is better.

— Italian proverb

3

Communication with Trust

Contents

3.1	Trust in $m\mathcal{A}^\rho$	65
3.1.1	<i>un/mis</i> -Trustworthy Announcement	66
3.1.2	Desired Properties	74
3.2	Capturing Trust with Update Models	77
3.2.1	$m\mathcal{A}^*$ <i>un</i> -Trustworthy Announcement	77
3.2.2	$m\mathcal{A}^*$ <i>mis</i> -Trustworthy Announcement	79

3.1 Trust in $m\mathcal{A}^\rho$

As already mentioned, multi-agent planning and epistemic reasoning have recently gained attention from several research communities. Efficient autonomous systems that can reason in these domains could lead to winning strategies in various fields such as economy [Aumann et al., 1995], security [Balliu et al., 2011], justice [Prakken, 2013], politics [Carbonell Jr, 1978] and can be exploited by autonomous devices, *e.g.*, self-driving cars, that can control several aspects of our daily life. We already explained why epistemic reasoners are not only interested in the state of the world, but also in the knowledge/beliefs of the agents. Nonetheless, the existing epistemic action languages [Bolander and Andersen, 2011, Muise et al., 2015, Baral et al., 2015, Fabiano et al., 2020, Baral et al., 2022] are able to model several

families of problems and study their information flows but cannot comprehensively reason on aspects like *trust*, *dishonesty*, *deception*, and other subtle epistemic concepts. To exploit epistemic reasoning in complex real-world scenarios it is, then, necessary to introduce the aforementioned epistemic nuances in the formalism used to express epistemic domains.

In this first section, we present an “expansion” of $m\mathcal{A}^p$ that allows us to formalize the notion of *Trust*. We do so by introducing two new actions that model the information exchange between agents when the idea of trust is involved:

(i) *un-trustworthy announcement*; and

(ii) *mis-trustworthy announcement*.

In particular, (i) *un-trustworthy announcement* formalizes the situation when the *untrusty* agents will not change their beliefs about the world no matter what the announcer says; and (ii) *mis-trustworthy announcement* captures the scenarios where the announcer, when not trusted, is believed to have a systematic faulty perception of the announced environment’s properties. This leads the untrusty agents to believe the opposite of what has been announced.

3.1.1 *un/mis-Trustworthy Announcement*

We are now ready to provide a formal definition of the actions *un-trustworthy announcement* and *mis-trustworthy announcement*. That is, an agent can or cannot trust what another agent is telling her/him and act consequently. The expressions

$$i \text{ t_announces } a \text{ if } \varphi;$$

and

$$i \text{ m_announces } a \text{ if } \varphi;$$

express that the agent i is executing an *un-trustworthy announcement* or a *mis-trustworthy announcement*, respectively.

In defining the actions let us consider a static and globally visible version of “trust” that can be formalized with a simple function $\mathcal{T} : \mathcal{AG} \times \mathcal{AG} \mapsto \{0, 1\}$ and that enriches the definition of MEP domain (Definition 3.1).

Definition 3.1: MEP Domain with Trust

A *MEP domain with Trust* is a tuple $D = \langle \mathcal{F}, \mathcal{AG}, \mathcal{A}, \mathcal{T}, \varphi_{ini}, \varphi_{goal} \rangle$, where the additional (with respect to Definition 1.15) element \mathcal{T} contains the trust relations between agents:

$$\mathcal{T}(i, j) = \begin{cases} 1 & \text{if } [j \text{ trust } i] \\ 0 & \text{otherwise} \end{cases}$$

where i and $j \in D(\mathcal{AG})$.

We will consider only the case where \mathcal{T} is a static and globally visible function. Let us notice that having \mathcal{T} to be dynamic is easily achievable. In particular, we just need to define how \mathcal{T} may vary, *e.g.*, making the function depending on some fluents value. For the sake of simplicity, let us imagine \mathcal{T} to be fixed and not dependent on the plan execution. On the other hand, making \mathcal{T} not globally visible—*i.e.*, each agent knows her/his own version of the *trust* function—is not straightforward. The problem arises when two agents have different views of the same trust relation leading to the generation of different e-states (*i.e.*, each agent must preserve its separate view of the domain). We leave the investigation of this scenario as future work.

To clarify the e-state update after the execution of the new actions we will also present a graphical representation of the transition function application. The examples of execution will be based on a variation of the *Grapevine* domain, firstly introduced by Kominis and Geffner [2015], described later in Planning Domain 3.1. Let us now provide a formal definition of e-state update of the two new actions of $m\mathcal{A}^p$.

***un*-Trustworthy Announcement**

We can now introduce the transition function for an *un*-trustworthy announcement. Intuitively, this action models an announcement where the listening agents can or cannot trust the announcer. That is:

- the *trusty* agents will update their belief consistently with what has been announced; and

- the *untrustworthy*¹ ones will maintain their beliefs about the world and will only update their perspective on the beliefs of the trustworthy agents.

Let us recall that the sets $\mathbf{F}_a, \mathbf{P}_a, \mathbf{O}_a$ represent the set of fully observant, partially observant, and oblivious agents with respect to the execution of an action instance $a\langle i \rangle$, respectively.

Let a domain D , its set of action instances $D(\mathcal{AI})$, and the set \mathcal{S} of all the possibilities reachable from $D(\varphi_{ini})$ with a finite sequence of action instances be given. The transition function $\Phi : D(\mathcal{AI}) \times \mathcal{S} \rightarrow \mathcal{S} \cup \{\emptyset\}$ for the *un-trustworthy* announcement relative to D is presented in Definition 3.2. Intuitively, this transition function allows, through the use of Υ and Ψ , to model the idea that the untrustworthy agents maintain their beliefs while knowing that the trustworthy ones updated their point of view of the “physical world” (and vice versa).

¹The agents that are fully observant with respect to the announcement but that do not trust the announcer.

Definition 3.2: $m\mathcal{A}^p$ un-Trustworthy Announcement

Allow us to use the compact notation $u(\mathcal{F}) = \{\mathbf{f} \mid \mathbf{f} \in D(\mathcal{F}) \wedge u \models \mathbf{f}\} \cup \{\neg\mathbf{f} \mid \mathbf{f} \in D(\mathcal{F}) \wedge u \not\models \mathbf{f}\}$ for the sake of readability. Let an action instance $\mathbf{a}(i) \in D(\mathcal{AI})$ where agent $i \in D(\mathcal{AG})$ announces the fluent formula ϕ , an and a possibility $u \in \mathcal{S}$ be given.

If \mathbf{a} is not executable in u , then $\Phi(\mathbf{a}, u) = \emptyset$ otherwise $\Phi(\mathbf{a}, u) = u'$, where:

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

$$u'(\mathcal{F}) = u(\mathcal{F})$$

$$u'(j) = \begin{cases} u(j) & \text{if } j \in \mathbf{O}_a \\ \bigcup_{w \in u(j)} \Upsilon(\mathbf{a}, w) \cup \Psi(\mathbf{a}, w) & \text{if } j \in \mathbf{P}_a \\ \bigcup_{w \in u(j)} \Upsilon(\mathbf{a}, w) & \text{if } j \in \mathbf{F}_a \wedge e(\mathbf{a}, u) = 1 \\ \bigcup_{w \in u(j)} \Psi(\mathbf{a}, w) & \text{if } j \in \mathbf{F}_a \wedge e(\mathbf{a}, u) = 0 \end{cases}$$

with $\Upsilon(\mathbf{a}, w) = w'$ such that

$$w'(\mathcal{F}) = w(\mathcal{F})$$

$$w'(j) = \begin{cases} w(j) & \text{if } j \in \mathbf{O}_a \\ \bigcup_{v \in w(j)} \Phi(\mathbf{a}, v) & \text{if } j \in \mathbf{P}_a \\ \bigcup_{v \in w(j)} \Upsilon(\mathbf{a}, v) & \text{if } j \in \mathbf{F}_a \wedge \mathcal{T}(j, i) = 0 \\ \bigcup_{v \in w(j): c(\mathbf{a}, v) = 1} \Upsilon(\mathbf{a}, v) & \text{if } j \in \mathbf{F}_a \wedge \mathcal{T}(j, i) = 1 \end{cases}$$

and $\Psi(\mathbf{a}, w) = w'$ such that

$$w'(\mathcal{F}) = w(\mathcal{F})$$

$$w'(j) = \begin{cases} w(j) & \text{if } j \in \mathbf{O}_a \\ \bigcup_{v \in w(j)} \Phi(\mathbf{a}, v) & \text{if } j \in \mathbf{P}_a \\ \bigcup_{v \in w(j)} \Psi(\mathbf{a}, v) & \text{if } j \in \mathbf{F}_a \wedge \mathcal{T}(j, i) = 0 \\ \bigcup_{v \in w(j): c(\mathbf{a}, v) = 0} \Psi(\mathbf{a}, v) & \text{if } j \in \mathbf{F}_a \wedge \mathcal{T}(j, i) = 1 \end{cases}$$

for each agent $j \in D(\mathcal{AG})$.

An Example of Execution As mentioned above, we will provide a graphical representation of the newly introduced transition function. Let us remember that

we will represent a possibility as a graph where the nodes correspond to the possible worlds while the edges encode the beliefs of the agents. The thicker node represents the pointed possibility. Moreover, to extract the point of view of the agents from a graph we need to follow the entailment rules (Definition 2.11) starting from the pointed possibility. In Planning Domain 3.1 we briefly describe the instance that will be used as a running example. Since we are only interested in showing how to e-state update works we will omit the actions and goal description.

Planning Domain 3.1: Trust-Grapevine

$n \geq 2$ agents are located in $k \geq 2$ rooms. Each agent knows $j \geq 0$ secrets. An agent can move freely to other rooms, and she/he can share a “secret” with the agents that are in the room with her/him. Moreover, the agents will be aware of the execution of announcements made in adjacent rooms without actually knowing the truth value of the announced fluent. Each agent can or cannot *trust* what another agent shares^a.

Our running example considers five agents: A, B, C, D, and E. Initially we have that:

- A, B, C are located in the same room (`room_1`) while D is in a room (`room_2`) adjacent to `room_1` and E is located in `room_3`, not adjacent to `room_1`;
- agents B and D trust A while C and E do not;
- only agent A knows `secret_a`;
- the value of `secret_a` is \top ; and
- initially everyone knows the position of each agent and that only A knows the value of `secret_a`.

In Figure 3.1 we present a graphical representation of the above described initial state.

^aLet us notice that since the idea of trust is involved, each agent, in order to learn a secret, needs to witness an announcement of agents that she/he trusts, making the newly presented domain slightly more intricate than the original Grapevine.

Figure 3.2 represents the e-state generated after the execution of the *un-trustworthy* announcement action instance `announce_secret_a(A)` (`ann_a` for brevity). With `ann_a`, A announces the value of `secret_a`. Let us note that from the position of the agents we know that $A, B, C \in \mathbf{F}_{\text{ann}_a}$, $D \in \mathbf{P}_{\text{ann}_a}$, and $E \in \mathbf{O}_{\text{ann}_a}$.

Once again, from Figure 3.2, we can derive that B—a trusty fully observant—believes `secret_a` to be true (\top) while C—an untrusty fully observant—did not change her/his direct belief about `secret_a`, but changed her/his beliefs on other agents’, *e.g.*, B, beliefs. More intricate relations, described in Proposition 3.1, can also be derived from Figure 3.2.

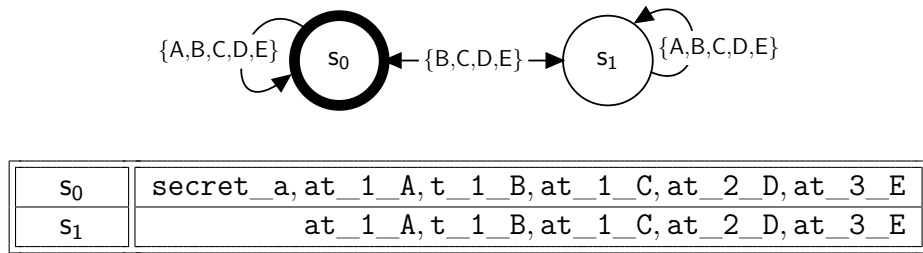


Figure 3.1: The initial e-state described in Planning Domain 3.1. The bottom Table presents the fluents interpretation of each possibility (as usual, only the positive fluents are reported).

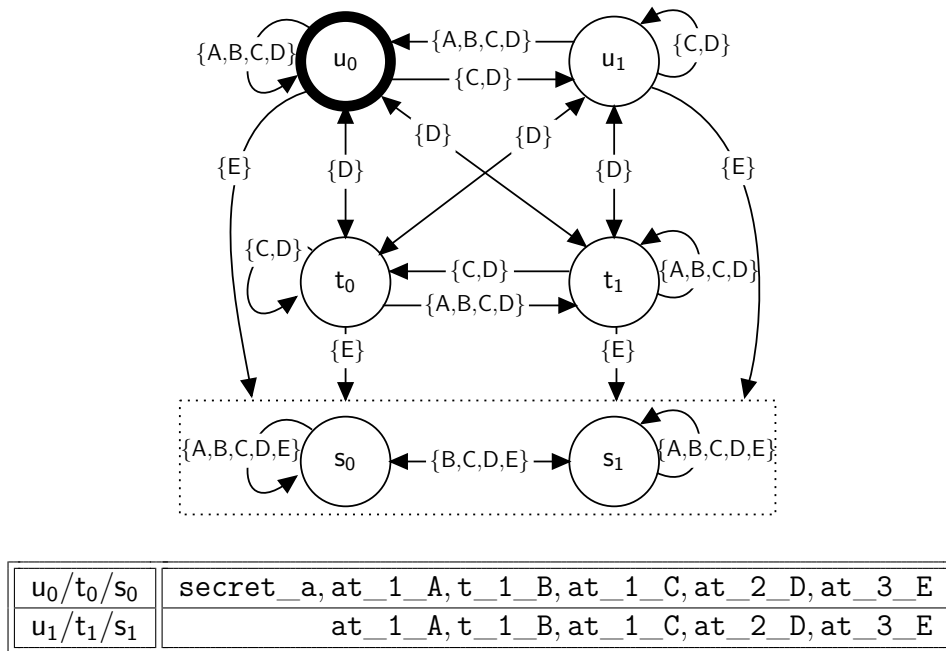


Figure 3.2: The e-state obtained after executing the *un-trustworthy* announcement action `ann_a` in the e-state represented in Figure 3.1.

***mis*-Trustworthy Announcement**

In Definition 3.2 we assumed that an agent j , when does not trust the announcer, will not change her/his beliefs about what has been announced. That is, an untrusty agent will not change her/his perspective on the “physical” state of the world. Let us notice that this type of trust captures the idea that, for the untrusty agents, the announcer is not reliable and the information she/he is providing is not worth taking into consideration as it can be not accurate.

Depending on the scenario it could be necessary to model a stronger concept of untrust. In particular, it could be required to design an *un*-trustworthy announcement such that the untrusty agents will believe the contrary of what has been announced (while still believing that the announcer believes what she/he announced). We will call this type of action *mis*-trustworthy announcement. The formalization of such variation is presented in Definition 3.3.

Let us note that the transition functions introduced in Definition 3.2 and Definition 3.3 only differ in the specification of Υ and Ψ for the untrusty fully observant agents. This difference is needed to represent the fact that in the case of an *un*-trustworthy announcement the untrusty agents maintain their beliefs while in the *mis*-trustworthy one they will believe the opposite of what has been announced.

Definition 3.3: $m\mathcal{A}^p$ *mis*-Trustworthy Announcement

Let an action instance $\mathbf{a}\langle i \rangle \in D(\mathcal{AI})$ where agent $i \in D(\mathcal{AG})$ announces the fluent formula ϕ and a possibility $\mathbf{u} \in \mathcal{S}$ be given.

If \mathbf{a} is not executable in \mathbf{u} , then $\Phi(\mathbf{a}, \mathbf{u}) = \emptyset$ otherwise $\Phi(\mathbf{a}, \mathbf{u}) = \mathbf{u}'$, where:

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

$$\mathbf{u}'(\mathcal{F}) = \mathbf{u}(\mathcal{F})$$

$$\mathbf{u}'(j) = \begin{cases} \mathbf{u}(j) & \text{if } j \in \mathbf{O}_a \\ \bigcup_{w \in \mathbf{u}(j)} \Upsilon(\mathbf{a}, w) \cup \Psi(\mathbf{a}, w) & \text{if } j \in \mathbf{P}_a \\ \bigcup_{w \in \mathbf{u}(j)} \Upsilon(\mathbf{a}, w) & \text{if } j \in \mathbf{F}_a \wedge e(\mathbf{a}, \mathbf{u}) = 1 \\ \bigcup_{w \in \mathbf{u}(j)} \Psi(\mathbf{a}, w) & \text{if } j \in \mathbf{F}_a \wedge e(\mathbf{a}, \mathbf{u}) = 0 \end{cases}$$

with $\Upsilon(\mathbf{a}, w) = w'$ such that

$$w'(\mathcal{F}) = w(\mathcal{F})$$

$$w'(j) = \begin{cases} w(j) & \text{if } j \in \mathbf{O}_a \\ \bigcup_{v \in w(j)} \Phi(\mathbf{a}, v) & \text{if } j \in \mathbf{P}_a \\ \bigcup_{v \in w(j): e(\mathbf{a}, v) = 0} \Upsilon(\mathbf{a}, v) & \text{if } j \in \mathbf{F}_a \wedge \mathcal{T}(j, i) = 0 \\ \bigcup_{v \in w(j): e(\mathbf{a}, v) = 1} \Upsilon(\mathbf{a}, v) & \text{if } j \in \mathbf{F}_a \wedge \mathcal{T}(j, i) = 1 \end{cases}$$

and $\Psi(\mathbf{a}, w) = w'$ such that

$$w'(\mathcal{F}) = w(\mathcal{F})$$

$$w'(j) = \begin{cases} w(j) & \text{if } j \in \mathbf{O}_a \\ \bigcup_{v \in w(j)} \Phi(\mathbf{a}, v) & \text{if } j \in \mathbf{P}_a \\ \bigcup_{v \in w(j): e(\mathbf{a}, v) = 1} \Psi(\mathbf{a}, v) & \text{if } j \in \mathbf{F}_a \wedge \mathcal{T}(j, i) = 0 \\ \bigcup_{v \in w(j): e(\mathbf{a}, v) = 0} \Psi(\mathbf{a}, v) & \text{if } j \in \mathbf{F}_a \wedge \mathcal{T}(j, i) = 1 \end{cases}$$

for each agent $j \in D(\mathcal{AG})$.

An Example of Execution As for the *un*-trustworthy announcement, we will provide an example of *mis*-trustworthy announcement execution. The initial state is identical to the one introduced in Planning Domain 3.1 . The only difference is that

now the action `announce_secret_a(A)` (or `ann_a` for brevity) is a *mis*-trustworthy announcement instead of an *un*-trustworthy announcement. The initial state is, therefore, represented in Figure 3.1 while the e-state obtained after the execution of the *mis*-trustworthy announcement is shown in Figure 3.3. From Figure 3.3, we can extrapolate that **B**—a trusty fully observant—believes `secret_a` to be true (\top) while **C**—an untrusty fully observant—believes the opposite, *i.e.*, `secret_a` = \perp . As for the previous actions, also for *mis*-trustworthy announcement more intricate relations, described in Proposition 3.2, can also be derived from Figure 3.3.

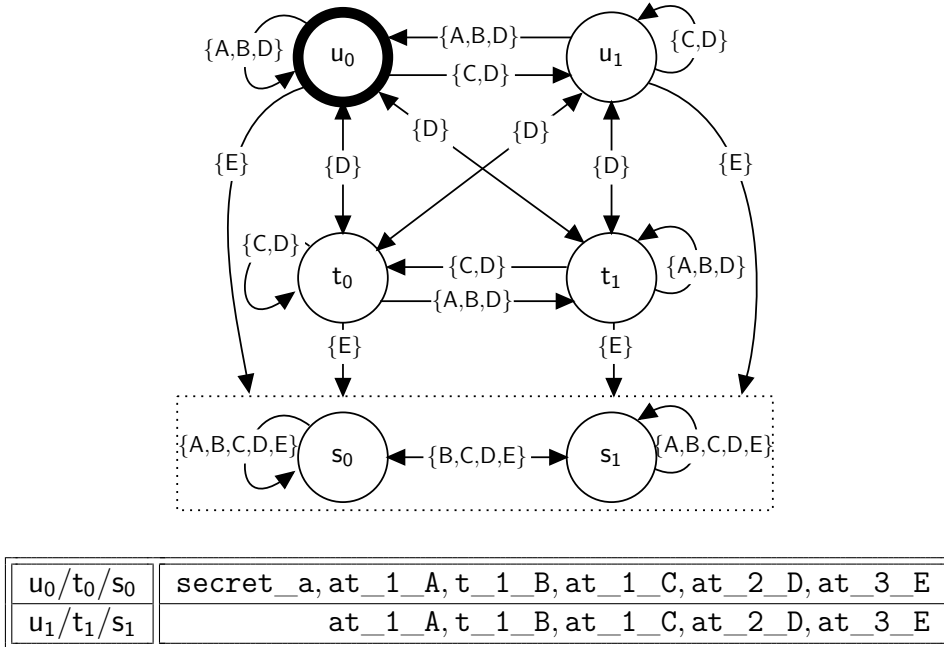


Figure 3.3: The e-state obtained after executing the *mis*-trustworthy announcement action `ann_a` in the e-state represented in Figure 3.1.

3.1.2 Desired Properties

In Section 2.2.2 some useful properties that correctly capture certain intuitions concerning the effects of the various types of actions in $m\mathcal{A}^p$ are listed. Similarly, in what follows, we will provide some properties that the e-state update, after executing the *un*/*mis*-trustworthy announcement, meets. In Appendix A.3 we will show the formal proofs that these properties hold. As usual, we will indicate the sets of partially observant and oblivious agents (with respect to the action

instance $\mathbf{a}\langle i \rangle$ with \mathbf{P}_a and \mathbf{O}_a , respectively. Moreover, we will indicate the set of trusty fully observant agents with \mathbf{F}_a while will indicate the set of untrusty fully observant with \mathbf{U}_a .

Proposition 3.1: *un-Trustworthy Announcement Properties*

Let $\mathbf{a}\langle i \rangle$ be an *un-trustworthy announcement* action instance where an agent i **t_**announces ϕ . Let \mathbf{e} be an e -state and let \mathbf{e}' be its updated version (that is, $\Phi(\mathbf{a}, \mathbf{e}) = \mathbf{e}'$), then in $m\mathcal{A}^\rho$ it holds that:

- (1) $\mathbf{e}' \models \mathbf{C}_{\mathbf{F}_a} \phi$;
- (2) $\mathbf{e}' \models \mathbf{C}_{\mathbf{U}_a} (\mathbf{C}_{\mathbf{F}_a} \phi)$;
- (3) $\mathbf{e}' \models \mathbf{C}_{\mathbf{P}_a} (\mathbf{C}_{\mathbf{F}_a} \phi \vee \mathbf{C}_{\mathbf{F}_a} \neg \phi)$;
- (4) $\mathbf{e}' \models \mathbf{C}_{\mathbf{F}_a \cup \mathbf{U}_a} (\mathbf{C}_{\mathbf{P}_a} (\mathbf{C}_{\mathbf{F}_a} \phi \vee \mathbf{C}_{\mathbf{F}_a} \neg \phi))$;
- (5) for every agent $\mathbf{y} \in \mathbf{O}_a$ and a belief formula φ , $\mathbf{e}' \models \mathbf{B}_y(\varphi)$ iff $\mathbf{e} \models \mathbf{B}_y(\varphi)$;
and
- (6) for every pair of agents $\mathbf{x} \in \mathbf{F}_a \cup \mathbf{U}_a \cup \mathbf{P}_a$ and $\mathbf{y} \in \mathbf{O}_a$ and a belief formula φ , if $\mathbf{e} \models \mathbf{B}_x(\mathbf{B}_y(\varphi))$ then $\mathbf{e}' \models \mathbf{B}_x(\mathbf{B}_y(\varphi))$.
- (7) for every agent $\mathbf{y} \in \mathbf{U}_a$, $\mathbf{e}' \models \mathbf{B}_y(\phi) / \mathbf{B}_y(\neg \phi) / (\neg \mathbf{B}_y(\phi) \wedge \neg \mathbf{B}_y(\neg \phi))$ iff $\mathbf{e} \models \mathbf{B}_y(\phi) / \mathbf{B}_y(\neg \phi) / (\neg \mathbf{B}_y(\phi) \wedge \neg \mathbf{B}_y(\neg \phi))$;

The properties presented in Proposition 3.1 capture some fundamental aspects of an *un-trustworthy announcement* action. Intuitively:

- (1) Captures the idea that all the trusty fully observant agents should believe *(i)* what has been announced; and *(ii)* that all the other trusty fully observant agents believe what has been announced and so on *ad infinitum* (that is why we use the **C** operator).
- (2) Models the fact that all the untrusty agents believe that all the trusty ones have common belief of what has been announced.
- (3) Captures that the partially observants believe that the trusty fully observants have common knowledge of what has been announced while the partially observants themselves do not know the announced value.

- (4) States that the fully observant agents have common knowledge of the previous property.
- (5) Captures the fact that the oblivious agents do not change their beliefs.
- (6) States that the observant agents (trusty, untrusty, and partial) believe that the oblivious agents did not change their beliefs.
- (7) Models the idea that all the untrusty agents do not modify their beliefs about the announced values.

As we did for the *un*-trustworthy announcement, let us identify some properties also for the *mis*-trustworthy announcement action.

Proposition 3.2: *mis*-Trustworthy Announcement Properties

Let $\mathbf{a}\langle i \rangle$ be a *mis*-trustworthy announcement action instance where an agent i **m_announces** ϕ . Let \mathbf{e} be an e -state and let \mathbf{e}' be its updated version (that is, $\Phi(\mathbf{a}, \mathbf{e}) = \mathbf{e}'$), then in $m\mathcal{A}^p$ Items (1) to (6) of Proposition 3.1 hold. In addition,

- (8) $\mathbf{e}' \models \mathbf{C}_{\mathbf{U}_a} \neg \phi$;
- (9) $\mathbf{e}' \models \mathbf{C}_{\mathbf{F}_a} (\mathbf{C}_{\mathbf{U}_a} \neg \phi)$;
- (10) $\mathbf{e}' \models \mathbf{C}_{\mathbf{P}_a} (\mathbf{C}_{\mathbf{U}_a} \phi \vee \mathbf{C}_{\mathbf{U}_a} \neg \phi)$;

Proposition 3.2 describes the core ideas behind a *mis*-trustworthy announcement action. While Items (1) to (6) of Proposition 3.1 have already been described, the intuitive meaning of the remaining ones is as follows.

- (8) Captures the idea that all the untrusty fully observant agents should believe (i) the contrary of what has been announced; and (ii) that all the other untrusty fully observant agents believe the negation of what has been announced and so on *ad infinitum* (that is why we use the **C** operator).
- (9) Models the fact that all the trusty agents believe that all the untrusty ones have common belief of the negation of what has been announced.

- (10) Captures that the partially observants believe that the untrusty fully observants have common belief of what has been announced, while the partially observant themselves do not know the announced value.

3.2 Capturing Trust with Update Models

Since most of the work in dynamic epistemic planning revolves around the concepts of Kripke structures and update models, *e.g.*, Bolander and Andersen [2011], Baral et al. [2022], we believe that formalizing the aforementioned actions using these concepts would provide interesting insights for the community. In this paragraph we, therefore, succinctly present the update models that define the announcement actions where the trust relation \mathcal{T} is taken into consideration.

3.2.1 $m\mathcal{A}^*$ *un-Trustworthy* Announcement

Let us begin by describing the update model related to the *un-trustworthy* announcement for the language $m\mathcal{A}^*$. We will follow the scheme presented by Baral et al. [2015, 2022] to introduce a new action's update model. The transition function is derived by applying the update model to a Kripke structure. Let us also recall that the sets $\mathbf{F}_a, \mathbf{P}_a, \mathbf{O}_a$ represent the set of fully observant, partially observant, and oblivious agents with respect to an action instance $a(i)$, respectively. Definition 3.4 presents formally the update model of an *un-trustworthy* announcement. Let us notice that an update instance of an *un-trustworthy* announcement action occurrence has five events to capture the idea of both updating the beliefs of the trusty agents and maintaining the beliefs of the untrusty ones. Figure 3.4 provides a graphical representation of this update template. Each event is associated with sets of states and edges where the truth value of the announced fluent formula is either known to be true, known to be false, or unknown.

Definition 3.4: $m\mathcal{A}^*$ *un-trustworthy announcement Update Model*

Let $\mathbf{a}\langle i \rangle$ be an *un-trustworthy announcement* action instance where agent i announces the fluent formula ϕ with the precondition ψ , a function $\mathcal{T} : \mathcal{AG} \times \mathcal{AG} \mapsto \{0, 1\}$ and a frame of reference $\rho = (\mathbf{F}_a, \mathbf{P}_a, \mathbf{O}_a)$. The update model for \mathbf{a} , \mathcal{T} , and ρ , $\omega(\mathbf{a}, \mathcal{T}, \rho)$, is defined by $\langle \Sigma, \mathcal{R}_1, \dots, \mathcal{R}_n \rangle$ where:

- $\Sigma = \{\sigma, \tau, \eta, \mu, \epsilon\}$;

- \mathcal{R}_j is defined by:

$$\mathcal{R}_j = \begin{cases} \{(\sigma, \sigma), (\tau, \tau), (\epsilon, \epsilon)\} & \text{if } i \in \mathbf{F}_a \wedge \mathcal{T}(j, i) = 1 \\ \{(x, x'), (y, y'), (\epsilon, \epsilon)\} & \text{if } i \in \mathbf{F}_a \wedge \mathcal{T}(j, i) = 0 \\ \text{where } x, x' \in \{\sigma, \eta\} \wedge y, y' \in \{\mu, \tau\} \\ \{(x, y), (\epsilon, \epsilon)\} & \text{if } j \in \mathbf{P}_a \\ \text{where } x, y \in \{\sigma, \tau, \eta, \mu\} \\ \{(x, \epsilon), (\epsilon, \epsilon)\} & \text{if } j \in \mathbf{O}_a \\ \text{where } x \in \{\sigma, \tau, \eta, \mu\} \end{cases}$$

- The preconditions are defined by:

$$pre(x) = \begin{cases} \psi \wedge \phi & \text{if } x \in \{\sigma, \mu\} \\ \psi \wedge \neg\phi & \text{if } x \in \{\eta, \tau\} \\ \top & \text{if } x = \epsilon \end{cases}$$

- *sub* is defined by $sub(x) = \emptyset$ for each $x \in \Sigma$.

- We identify σ as the *pointed event*.

The update instance for the *un-trustworthy announcement* action occurrence \mathbf{a} and the frame of reference ρ is $(\omega(\mathbf{a}, \mathcal{T}, \rho), \{\sigma, \tau, \eta, \mu\})$.

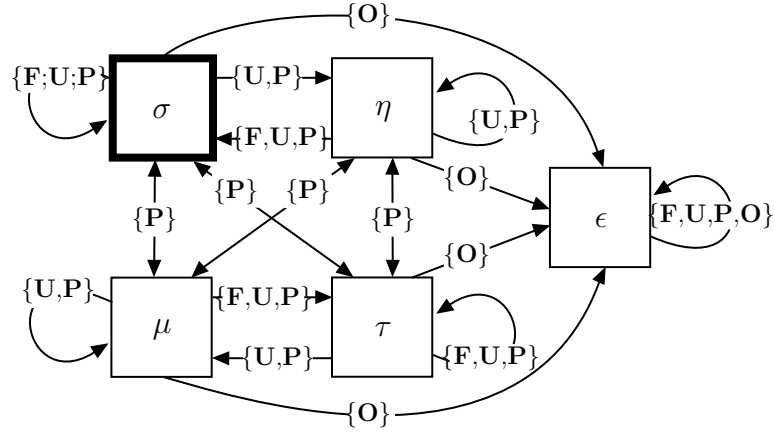


Figure 3.4: The update template (Σ, σ) for the *un-trustworthy* announcement. \mathbf{F} , \mathbf{U} , \mathbf{P} , \mathbf{O} represent the trusty fully observant, untrusty fully observant, partially observant, and oblivious agents respectively.

3.2.2 $m\mathcal{A}^*$ *mis-Trustworthy* Announcement

Let us now introduce the transition function related to the *mis-trustworthy* announcement action. As before, we will follow the scheme of Baral et al. [2015, 2022] to introduce a new action's update model, and the sets \mathbf{F}_a , \mathbf{P}_a , \mathbf{O}_a represent the set of fully observant, partially observant, and oblivious agents with respect to an action instance $\mathbf{a}\langle i \rangle$, respectively.

Let us note that Definition 3.5 only varies, with respect to Definition 3.4, in the behavior of \mathcal{R}_j for the untrusty fully observant agents (*i.e.*, $j \in \mathbf{F}_a \wedge \mathcal{T}(j, i) = 0$).

Definition 3.5: $m\mathcal{A}^*$ *mis*-trustworthy announcement Update Model

Let $\mathbf{a}\langle i \rangle$ be an *mis*-trustworthy announcement action instance where agent i announces the fluent formula ϕ with the precondition ψ , a function $\mathcal{T} : \mathcal{AG} \times \mathcal{AG} \mapsto \{0, 1\}$ and a frame of reference $\rho = (\mathbf{F}_a, \mathbf{P}_a, \mathbf{O}_a)$. The update model for \mathbf{a} , \mathcal{T} , and ρ , $\omega(\mathbf{a}, \mathcal{T}, \rho)$, is defined by $\langle \Sigma, \mathcal{R}_1, \dots, \mathcal{R}_n \rangle$ where:

- $\Sigma = \{\sigma, \tau, \eta, \mu, \epsilon\}$;

- \mathcal{R}_i is defined by:

$$\mathcal{R}_j = \begin{cases} \{(\sigma, \sigma), (\tau, \tau), (\epsilon, \epsilon)\} & \text{if } i \in \mathbf{F}_a \wedge \mathcal{T}(j, i) = 1 \\ \{(\eta, \eta), (\mu, \mu), (\epsilon, \epsilon)\} & \text{if } i \in \mathbf{F}_a \wedge \mathcal{T}(j, i) = 0 \\ \text{where } x, x' \in \{\sigma, \eta\} \wedge y, y' \in \{\mu, \tau\} \\ \{(x, y), (\epsilon, \epsilon)\} & \text{if } j \in \mathbf{P}_a \\ \text{where } x, y \in \{\sigma, \tau, \eta, \mu\} \\ \{(x, \epsilon), (\epsilon, \epsilon)\} & \text{if } j \in \mathbf{O}_a \\ \text{where } x \in \{\sigma, \tau, \eta, \mu\} \end{cases}$$

- The preconditions are defined by:

$$pre(x) = \begin{cases} \psi \wedge \phi & \text{if } x \in \{\sigma, \mu\} \\ \psi \wedge \neg\phi & \text{if } x \in \{\eta, \tau\} \\ \top & \text{if } x = \epsilon \end{cases}$$

- *sub* is defined by $sub(x) = \emptyset$ for each $x \in \Sigma$.

- We identify σ as the *pointed event*.

The update instance for the *mis*-trustworthy announcement action occurrence \mathbf{a} and the frame of reference ρ is $(\omega(\mathbf{a}, \mathcal{T}, \rho), \{\sigma, \tau, \eta, \mu\})$.

The update template for *mis*-trustworthy announcement is presented in Figure 3.5.

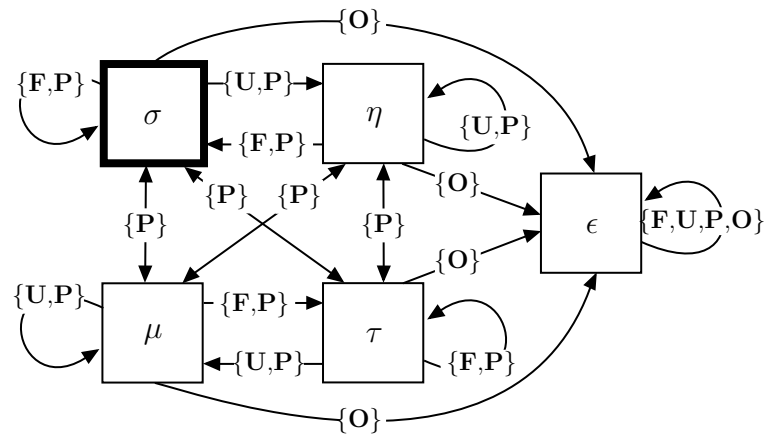


Figure 3.5: The update template (Σ, σ) for the *mis*-trustworthy announcement. **F**, **U**, **P**, **O** represent the trusty fully observant, untrusty fully observant, partially observant, and oblivious agents respectively.

People get built different. We don't need to figure it out, we just need to respect it.

— Princess Bubblegum
in “*Bonnie and Neddy*”
Adventure Time

4

Trust, Misconception, and Lies in MEP

Contents

4.1	Agents' Attitudes and Inconsistent Beliefs	83
4.1.1	Enriched Domains	85
4.2	Updated Transition Function	87
4.2.1	Examples of Actions Execution	92
4.2.2	Desired Properties	99
4.3	Related Work	100

4.1 Agents' Attitudes and Inconsistent Beliefs

In the previous chapter, we presented a formalization for announcements when the concept of trust is taken into consideration. While this introduction expands the range of scenarios that $m\mathcal{A}^p$ may represent, it still does not fully capture aspects like dishonesty, deception, and incongruent beliefs.

In this chapter, we further enrich the language $m\mathcal{A}^p$ with the concept of agents' *attitudes*. Our idea of attitudes stems from the concept of *dynamic attitudes* that “*represent the agent's assessment of the reliability of the source*” introduced by Rodenhäuser [2014]. We define basic attitudes that capture how an agent reacts when another agent is informing her/him about something. In the real world, in fact, it is often the case that we associate an idea of reliability with

an information source. This work captures this idea by having agents behave accordingly to the following attitudes: *doubtful*, *impassive*, *trustful*, *mistrustful*, or *stubborn* (detailed descriptions are given later).

In addition to the agents' relation with the information source, we also consider the scenario when agents learn a fact that discords with their previous beliefs. When such a discrepancy arises, we talk about *inconsistent belief*. Since, as said in Chapter 2, we consider **KD45**_n-states, inconsistencies are relative only to the beliefs of an agent (and not to the actual world). Let us assume that agent *i* believes that $\neg\varphi$ is the case in the e-state u (*i.e.*, $u \models \mathbf{B}_i(\neg\varphi)$); in $m\mathcal{A}^p$ there are two main sources of inconsistencies: (i) *i* observes the real world—performing a *sensing* action—and learns φ (the opposite of what she/he believed); (ii) *i* learns φ as a result of an *announcement* performed by another agent *j*. In both scenarios, we must account for the belief of *i* after the action execution. In particular, the resulting e-state u' must be consistent with the axiom **D** of Table 1.1. In the former case, (i) we resolve the inconsistency by having *i* believing φ ; *i.e.*, we make sure that $u' \models \mathbf{B}_i(\varphi)$. This is a reasonable solution, as we assume that agents trust their senses when observing the world. In the latter, (ii) we must take into account the *attitude* of the agent with respect to the announcer *j*. As said by Rodenhäuser [2014], “*we are not only interested in the acceptance of new information (based on trust), but also in its rejection (based on distrust)*”. For instance, the listener may be skeptical or credulous, and thus she/he would change her/his belief according to her/his attitude. Let us notice that inconsistent beliefs are different from *false beliefs*. An agent has a false belief about a property φ if she/he believes φ to be true even if such property does not hold in the actual world. False beliefs are already allowed in $m\mathcal{A}^p$ as a result of the presence of oblivious agents in action instances.

Going back to the attitudes of agents, the notion of *trust* naturally arises. It is reasonable to assume that the listener *i* believes the announcer *j* if *i* trusts *j*. In particular, let us consider three attitudes¹ for fully observant agents that listen to an announcement: *trustful*, *mistrustful*, and *stubborn*. *Trustful* agents believe

¹We only consider basics attitudes, we leave the exploration of more complex ones as future work.

what the announcer tells them; *mistrustful* agents believe the opposite of what is announced; and *stubborn* agents do not modify their beliefs. Considering the case of *semi-private* announcements, we need to introduce the concept of attitude for partially observant agents as well. Specifically, we consider *impassive* and *doubtful* agents. *Impassive* agents keep their current beliefs, while *doubtful* agents believe neither what is being announced nor the opposite, regardless of their previous beliefs. Note that *stubborn* and *impassive* agents are different, as the former kind is aware of what is being announced—*i.e.*, the truth value of the property φ . Let us note that such attitudes are named to capture our personal idea of the behavior they represent and they are not meant to wholly describe the nuances of complex social attitudes such as, for example, stubbornness.

When communicating with their peers, agents might announce something that is *false* relative to their own point of view. We call *lies* such announcements. Similar to the notion of inconsistent belief, the truthfulness of announcements depends on the point of view of the announcer i —*i.e.*, i truthfully announces φ iff $u \models \mathbf{B}_i(\varphi)$.

Introducing these novel concepts enriches $m\mathcal{A}^\rho$ in terms of what the actions may describe/perform. The language can describe a much broader set of real-life scenarios where different degrees of trust relations between agents are needed. In summary, in this chapter we present, to the best of our knowledge, the first transition function that can update an *epistemic state*—*i.e.*, the knowledge/belief-graph of the agents—when considering: (i) *inconsistent beliefs*, *i.e.*, discrepancies between the beliefs currently held by an agent and some new information that she/he acquires; (ii) *trust* relations between agents; and (iii) the possibility for an agent to *lie*. In the next section, we formally incorporate such features in the semantics of $m\mathcal{A}^\rho$.

4.1.1 Enriched Domains

Let us start by providing some definitions necessary to introduce the updated transition function for $m\mathcal{A}^\rho$. In particular, let us formally introduce the idea of *frame of reference* in Definition 4.1, the concept of *attitude* in Definition 4.2 and finally the MEP domain enriched with attitudes in Definition 4.3. In what

follows, when clear from the context, we will make use of \mathbf{a} to indicate the action instance $\mathbf{a}\langle\alpha\rangle$, with $\alpha \subseteq D(\mathcal{AG})$.

Definition 4.1: Frame of Reference [Baral et al., 2015]

The *frame of reference* of an action instance $\mathbf{a}\langle\alpha\rangle$ is a partition $\rho_{\mathbf{a}\langle\alpha\rangle} = \langle \mathbf{F}_a, \mathbf{P}_a, \mathbf{O}_a \rangle$ of the set $D(\mathcal{AG})$, denoting the **F**ully observant, **P**artially observant, and **O**blivious agents of $\mathbf{a}\langle\alpha\rangle$, respectively.

The concept of attitude is strictly related to announcements. Therefore, in what follows, $\mathbf{a}\langle\alpha\rangle$ is assumed to be an announcement action. We recall that announcement action instances are assumed to have a single executor ($|\alpha| = 1$), referred to as the *announcer*. In this case, we make use of the short notation $\mathbf{a}\langle j \rangle$ in place of $\mathbf{a}\langle \{j\} \rangle$.

Definition 4.2: Attitude

The *attitude* of an agent determines how she/he updates her/his beliefs when new information is announced. Attitudes induce a refined partition of the frame of reference $\rho_{\mathbf{a}\langle j \rangle} = \langle \mathbf{F}_a, \mathbf{P}_a, \mathbf{O}_a \rangle$ as follows:

- $\mathbf{F}_a = \{j\} \cup \mathbf{T}_a \cup \mathbf{M}_a \cup \mathbf{S}_a$: fully observant agents may be *the executor*, *Trustful*, *Mistrustful*, or *Stubborn*;
- $\mathbf{P}_a = \mathbf{I}_a \cup \mathbf{D}_a$: partially observant agents may be *Impassive* or *Doubtful*.

Attitudes are specified with $m\mathcal{A}^p$ statements of the form “**has_attitude i wrt j att if φ** ” (where **att** is one of the attitudes of Definition 4.2) and they define the *trust relations* among agents. Such a statement asserts that i bears the attitude **att** towards j if the condition φ is met. We assume that the attitudes of the agents are publicly visible, *except for the attitude that the announcer has with respect to her/him-self*. That is, the announcer knows whether she/he is being truthful, lying or announcing something that she/he is unaware of, while other agents do not. Instead, trustful and stubborn agents believe that the announcer is truthful (*i.e.*, they believe that the executor believes the announced property), whereas mistrustful agents believe the announcer to be lying (*i.e.*, they believe that the announcer believes the negation of the announced property). In what follows we

assume this schema of trust with respect to the executor, although it can be easily adapted to best represent different scenarios. Finally, we assume that the announcer does not modify her/his own beliefs about the property being announced. The considered attitudes provide agents with a simple set of possible behaviors. More sophisticated attitudes can be built upon the ones introduced in Definition 4.2.

Definition 4.3: MEP Domain with Attitudes

A *MEP domain with attitudes* is a tuple $D = \langle \mathcal{F}, \mathcal{AG}, \mathcal{A}, \mathcal{AT}, \varphi_{ini}, \varphi_{goal} \rangle$, where the additional element \mathcal{AT} contains the attitudes relations of agents:

$$\mathcal{AT} = \{(i, j, \text{att}, \varphi) \mid [\text{has_attitude } i \text{ wrt } j \text{ att if } \varphi]\}.$$

4.2 Updated Transition Function

In this section, we provide a formalization of the transition function Φ of $m\mathcal{A}^\rho$ that captures the aspects that we previously discussed in this section. Let a MEP domain with attitudes $D = \langle \mathcal{F}, \mathcal{AG}, \mathcal{A}, \mathcal{AT}, \varphi_{ini}, \varphi_{goal} \rangle$, an agent $j \in D(\mathcal{AG})$, an e-state $\mathbf{u} \in D(\mathcal{S})$, and an action instance $\mathbf{a} \in D(\mathcal{AI})$ be given. The frame of reference $\rho_{\mathbf{a}}$ and the attitudes of the agents are determined by confronting the elements of the attitudes relation \mathcal{AT} with the possibility \mathbf{u} . If \mathbf{a} is not executable in \mathbf{u} , then $\Phi(\mathbf{a}, \mathbf{u}) = \emptyset$. Otherwise, we distinguish between ontic and epistemic actions.

Ontic Actions Since ontic actions are not affected by the introduction of inconsistent beliefs, or attitudes, the previous formalization described in Definition 2.12 is maintained.

Epistemic Actions Sensing and announcement actions modify the beliefs of agents. Since agents might acquire information that discords with previous beliefs, we must resolve the discrepancies. In the case of sensing actions, we consider all fully observant agents as executors. Since each agent trusts her/his senses, we have $\mathbf{F}_{\mathbf{a}} = \mathbf{T}_{\mathbf{a}}$. Similarly, we assume partially observant agents to keep their beliefs about

the physical features of the world unchanged, *i.e.*, $\mathbf{P}_a = \mathbf{I}_a$. Hence, the refined frame of reference of sensing actions is $\rho_{a\langle\mathbf{T}_a\rangle} = \langle\mathbf{T}_a, \mathbf{I}_a, \mathbf{O}_a\rangle$.

In the case of announcement actions, it is necessary to state both the executor $j \in D(\mathcal{AG})$ and the attitudes to resolve inconsistent beliefs. Therefore, the frame of reference of announcement actions is $\rho_{a\langle j\rangle} = \langle(\{j\}, \mathbf{T}_a, \mathbf{M}_a, \mathbf{S}_a), (\mathbf{I}_a, \mathbf{D}_a), \mathbf{O}_a\rangle$. During the computation of the update, the attitude of the announcer j is set to match the perspective of the agent being currently handled by the transition function. In particular, as mentioned before, the announcer considers her/him-self stubborn; given that the announcement does not intact her/his beliefs about the truth value of the announced property. On the other hand, trustful and stubborn agents consider the announcer to be truthful; and mistrustful agents consider the announcer to be lying. Notice that the announcer is aware of the perspectives of others on her/his attitude, and so are the remaining agents. Assuming private points of view on the agents' attitudes brings an extra overhead to the problem and, therefore, we will address this issue in future works.

Let us note that the actions' effects are assumed to be deterministic. This assumption can be relaxed, as shown by Kuter et al. [2008]. Nonetheless, this would generate a significant performance overhead that would render the planning process unfeasible, most of the time. Given that the interest of the epistemic planning community lies in trying to capture the agents' information relations, we leave the formalization of non-deterministic actions as future work.

We assume the presence of a unique statement that describes the effects of an epistemic action. Namely, we allow to sense/announce a single literal at a time. Therefore, we assume the presence of a unique fluent literal that describes the effects of epistemic actions to further avoid non-determinism. This limitation is necessary as the presented transition function of epistemic actions considers the negation of the effects that, if defined as a conjunction, would generate a disjunctive form (*i.e.*, non-deterministic effects). As mentioned above, we decided to avoid non-determinism considering the already poor scalability of MEP problems even without it. Nonetheless, relaxing this restriction would allow to represent domains in which

sensing/announcing fluent formulae might be central. Conversely, ontic actions are not subject to this restriction and, therefore, can affect conjunctions of literals.

Let ℓ be the (unique) fluent literal such that $[\mathbf{a}$ senses/announces $\ell] \in D$. With a slight abuse of notation, we define the *value* of ℓ in a possibility \mathbf{w} as $val(\mathbf{a}, \mathbf{w}) = \mathbf{w}(\ell)$. The *effect* $e(\mathbf{a})$ of action \mathbf{a} is equal to 1 if ℓ is a positive fluent literal ($e(\mathbf{a}) = 0$, otherwise). We use the following simplifications: given a possibility \mathbf{p} , (i) \mathbf{p}' denotes the updated version of \mathbf{p} ; and (ii) if not stated otherwise, we consider $\mathbf{p}'(\mathcal{F}) = \mathbf{p}(\mathcal{F})$. For clarity, we briefly describe each component of the transition function after its definition.

Definition 4.4: Epistemic Actions with Attitude in $m\mathcal{A}^p$

Let i be an agent (*i.e.*, $i \in D(\mathcal{AG})$). Applying an epistemic action instance \mathbf{a} on the pointed possibility \mathbf{u} results in the updated pointed possibility $\Phi(\mathbf{a}, \mathbf{u}) = \mathbf{u}'$ such that:

$$\mathbf{u}'(i) = \begin{cases} \mathbf{u}(i) & \text{if } i \in \mathbf{O}_{\mathbf{a}} \\ \mathbf{P}(\mathbf{a}, \mathbf{u}) & \text{if } i \in \mathbf{P}_{\mathbf{a}} \\ \mathbf{F}(\mathbf{a}, \mathbf{u}, 1) & \text{if } i \in \mathbf{T}_{\mathbf{a}} \\ \mathbf{F}(\mathbf{a}, \mathbf{u}, 0) & \text{if } i \in \mathbf{M}_{\mathbf{a}} \\ \mathbf{S}(\mathbf{a}, \mathbf{u}, e(\mathbf{a}), 1) & \text{if } i \in \mathbf{S}_{\mathbf{a}} \\ \mathbf{S}(\mathbf{a}, \mathbf{u}, e(\mathbf{a}), 0) & \text{if } i = j \end{cases}$$

where $\mathbf{P}, \mathbf{F}, \mathbf{S}$ are defined below.

DESCRIPTION: Φ modifies the beliefs of each agent on the announced fluent with respect to her/his attitude. For instance, the beliefs of trustful agents are updated by the sub-function \mathbf{F} . Each sub-function ($\mathbf{P}, \mathbf{F}, \mathbf{S}$) updates the nested beliefs of the agents, *i.e.*, the beliefs that the agents have of others' perspectives.

Helper functions χ and $\bar{\chi}$

We first define the *helper functions* χ and $\bar{\chi}$. Let $\mathbf{w}'_{\mathbf{x}} = \chi(\mathbf{a}, \mathbf{w}, \mathbf{x})$ and $\bar{\mathbf{w}}'_{\mathbf{x}} = \bar{\chi}(\mathbf{a}, \mathbf{w}, \bar{\mathbf{x}})$ where: (i) $\mathbf{w}'_{\mathbf{x}}$ and $\bar{\mathbf{w}}'_{\mathbf{x}}$ represent the possibility \mathbf{w} updated with χ and $\bar{\chi}$, respectively; (ii) \mathbf{x} and $\bar{\mathbf{x}}$ represent opposite Boolean values s.t. $\mathbf{x} = \neg\bar{\mathbf{x}}$; and (iii) let b be 1 and 0 when executing χ and $\bar{\chi}$, respectively. Then $\mathbf{w}'_{\mathbf{x}}$ and $\bar{\mathbf{w}}'_{\mathbf{x}}$ are defined as follows:

$$\mathbf{w}'_{\mathbf{x}}(\ell) = \begin{cases} \mathbf{x} & \text{if } \ell = \mathbf{f} \\ \mathbf{u}(\ell) & \text{otherwise} \end{cases} \quad \bar{\mathbf{w}}'_{\mathbf{x}}(\ell) = \begin{cases} \bar{\mathbf{x}} & \text{if } \ell = \mathbf{f} \\ \mathbf{u}(\ell) & \text{otherwise} \end{cases}$$

$$\left. \begin{array}{l} w'_x(i) \\ \bar{w}'_x(i) \end{array} \right\} = \begin{cases} w(i) & \text{if } i \in \mathbf{O}_a \\ P(a, w) & \text{if } i \in \mathbf{P}_a \\ \bigcup_{v \in w(i)} \chi(a, v, x) & \text{if } i \in \mathbf{T}_a \vee (i = j \wedge = 1) \\ \bigcup_{v \in w(i)} \bar{\chi}(a, v, \bar{x}) & \text{if } i \in \mathbf{M}_a \vee (i = j \wedge = 0) \\ S(a, w, x, 1) & \text{if } i \in \mathbf{S}_a \end{cases}$$

DESCRIPTION: Functions χ and $\bar{\chi}$ recursively update the nested beliefs of trustful and mistrustful agents (the cases of other attitudes are delegated to the respective sub-functions). However, they do not correspond to *trustful* and *mistrustful* attitudes. The functions χ and $\bar{\chi}$ are exploited by P and F/S by specifying the correct value of x to guarantee the correct update of the beliefs of partially and fully observant agents, respectively. We make use of two Boolean variables: (i) x encodes the truth value of ℓ believed by i ; (ii) b is a flag that keeps track of whether i is trustful ($b = 1$) or mistrustful ($b = 0$) with respect to the announcer.

Sub-function P

Let $w'_p = P(a, w)$. Then:

$$w'_p(i) = \begin{cases} w(i) & \text{if } i \in \mathbf{O}_a \\ \bigcup_{v \in w(i)} P(a, v) & \text{if } i \in \mathbf{I}_a \\ \bigcup_{v \in w(i)} \chi(a, v, 0) \cup \chi(a, v, 1) & \text{if } i \in \mathbf{D}_a \\ \bigcup_{v \in w(i)} \chi(a, v, val(a, v)) & \text{if } i \in \mathbf{T}_a \cup \mathbf{M}_a \cup \{j\} \\ \bigcup_{v \in w(i)} S(a, v, val(a, v), 1) & \text{if } i \in \mathbf{S}_a \end{cases}$$

DESCRIPTION: Function P updates the beliefs of partially observant agents. It updates their “direct beliefs” (*i.e.*, that represent their point of view) on ℓ and the nested beliefs of fully observant agents (by calling χ with $x = val(a, w)$). This guarantees that agents in \mathbf{P}_a believe that (mis)trustful agents are aware of the action’s effect. For doubtful agents χ is executed with both $x = 0$ and $x = 1$, forcing them to be ignorant about ℓ .

Sub-function F

Let $w'_f = F(a, w, b)$. Then:

$$w'_f(i) = \begin{cases} w(i) & \text{if } i \in \mathbf{O}_a \\ P(a, w) & \text{if } i \in \mathbf{P}_a \\ \bigcup_{v \in w(i)} \chi(a, v, e(a)) & \text{if } i \in \mathbf{T}_a \vee (i = j \wedge = 1) \\ \bigcup_{v \in w(i)} \bar{\chi}(a, v, \neg e(a)) & \text{if } i \in \mathbf{M}_a \vee (i = j \wedge = 0) \\ \bigcup_{v \in w(i)} S(a, v, e(a), 1) & \text{if } i \in \mathbf{S}_a \end{cases}$$

DESCRIPTION: Function **F** updates the point of views on ℓ of trustful and mistrustful agents, calling χ and $\bar{\chi}$, respectively. Moreover, **F** deals with the beliefs of other agents with respect to (mis)trustful agents. The flag b keeps track of whether **F** is executed from the perspective of a trustful ($b = 1$) or a mistrustful ($b = 0$) agent allowing to update i 's perspective on the beliefs of the announcer.

Sub-function S

Let $w'_s = S(a, w, x, s)$. Then:

$$w'_s(i) = \begin{cases} w(i) & \text{if } i \in \mathbf{O}_a \\ P(a, w) & \text{if } i \in \mathbf{P}_a \\ \bigcup_{v \in w(i)} \chi(a, v, x) & \text{if } i \in \mathbf{T}_a \vee (i = j \wedge s = 1) \\ \bigcup_{v \in w(i)} \bar{\chi}(a, v, \neg x) & \text{if } i \in \mathbf{M}_a \\ \bigcup_{v \in w(i)} S(a, v, x, s) & \text{if } i \in \mathbf{S}_a \vee (i = j \wedge s = 0) \end{cases}$$

DESCRIPTION: Function **S** keeps the “direct” beliefs of the executor and stubborn agents unchanged and it updates their perspective on other agents' beliefs. Here, we make use of two Boolean variables: (i) x is defined as in $\chi/\bar{\chi}$; (ii) s is used to identify whether the function has been called by a stubborn agent ($s = 1$) or if it is updating the “direct” beliefs of the executor ($s = 0$).

While Definition 4.4 formally defines how an e-state is updated after the execution of an epistemic action when agents' attitudes are considered, let us present its intuitive meaning. Let a be an announcement action (a sensing action can be thought of as a special case of an announcement). The point of view of oblivious agents remains untouched. Since a is an epistemic action, the fluents of the pointed world u' are unchanged with respect to its previous version u . On the other hand, trustful agents' points of view are changed to fit the announced property ℓ ; mistrustful

agents believe the opposite of what is announced; stubborn and impassive agents do not change their belief on ℓ . The perspective of doubtful agents is built by including also the opposite point of view with respect to ℓ . Higher-order beliefs are also correctly updated as stated in Proposition 4.1. Fully observant agents are aware of how each agent updates her/his beliefs. That is, they update the information states of other agents according to the attitudes of the other: (i) impassive agents do not change their belief on ℓ ; (ii) doubtful agents no longer hold any belief on ℓ ; and (iii) fully observant agents change their belief on ℓ as described above.

The information states of partially observant agents are updated similarly. The main difference lies in the fact that they are not aware of how fully observant agents update their points of view. Hence, partially observant agents maintain their perspective on the beliefs of fully observant agents unchanged.

Finally, the announcer considers her/him-self stubborn, since the announcement does not intact her/his beliefs, while other agents derive the attitude of the announcer depending on their own. As mentioned before, trustful and stubborn agents consider the announcer to be truthful, while mistrustful agents consider the announcer to be lying. Notice that the announcer is aware of the other agents' perspective on her/his attitude.

4.2.1 Examples of Actions Execution

In this section, we will show some examples of execution to better illustrate how the newly introduced transition function works. When needed we will describe the agents' attitudes. We will present, through labeled graphs, the e-state before and after the update for each example. While to capture all the combinations of attitudes we would need a far larger number of examples, we decided to provide only those that show the fundamental attitudes behavior and interactions. All the examples will be based on a simple variation of the Coin in the Box domain. Before presenting the examples of execution let us introduce the *Rigged Coin in the Box* in Planning Domain 4.1.

Planning Domain 4.1: Rigged Coin in the Box

Five agents, l, m, r, s, c , are inside a room containing a box. Agents l, m stand to the left of the box; r, s are at the box's right, and c is positioned in front of the box. Inside the box, agent c placed a rigged coin. Any agent might **peek** (*sensing* action) inside the box to learn the coin position. Since the coin is rigged, the actual position (either tails or heads up) is only visible when an agent is standing in front of the box (*i.e.*, c) or at its right (*i.e.*, r and s). On the other hand, any agent that stands at the box's left (*i.e.*, l and m) will always see the coin facing tails up. All the agents can share information through the action **announce**(*position*) (*announcement* action) that allows them to announce the position of the coin.

For the sake of readability, in all the following examples, we will use the same initial e-state while varying the agents' attitudes and the executed action. Let us now explain the initial configuration in Example 4.1 and then illustrate the corresponding e-state, in Figure 4.1, that from now on will be identified with u . A reminder on how to "read" an e-state graphical representation is presented right after the following initial e-state description.

Example 4.1: The Initial Configuration In our initial configuration we assume that is common belief that agents l, m, r , and s do not know the coin position, *i.e.*, $C_{D(\mathcal{AG})}(\neg B_i(\text{heads}) \wedge \neg B_i(\neg \text{heads}))$ with $i \in \{l, m, r, s\}$. On the other hand, agent c is aware of the coin position and the other agents know this, that is, in our initial e-state it holds $C_{D(\mathcal{AG})}(B_c(\text{heads}) \vee B_c(\neg \text{heads}))$. Assuming that the coin position is heads up the graphical representation of this e-state is as follows.

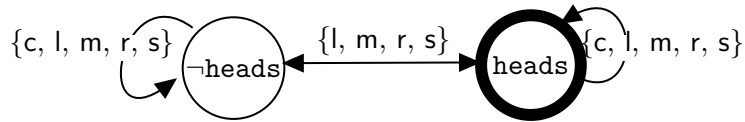


Figure 4.1: The initial e-state u of Planning Domain 4.1.

Before exploring the examples, let us briefly recall how to interpret the graphical representation of e-states. Consider Figure 4.1. The *bold-lined* world represents the actual world. If a world u is connected by an edge labeled with agent i to a world v , this means that in the world u agent i believes v to be possible.

In the initial state, each agent, except c , admits both the worlds where **heads** holds and where \neg **heads** holds. This means that such agents are uncertain about the coin position. On the other hand, agent c does know the actual configuration of the coin. We can understand this because in the actual world c admits only the world where **heads** hold.

Finally, observe that the remaining agents do not know what c knows. In fact, the formula $\mathbf{B}_m(\mathbf{B}_c(\mathbf{heads})) \vee \mathbf{B}_c(\neg\mathbf{heads})$ is true. On the other hand, the formula $\mathbf{B}_m(\mathbf{B}_c(\mathbf{heads}))$ does not hold in the initial state.

Example 4.2: Correct Sensing This example shows how u is updated after the execution of the action instance $\text{peek}\langle\{r, s\}\rangle$. As said in Planning Domain 4.1 both the agents r and s are able to correctly determine whether the coin lies tails or heads up. Since we are executing a sensing action we are only interested in defining the oblivious, the fully, and the partially observant agents. In particular, for this action instance, we assume r and s to be fully observant, l and m to be partially observant and c to be oblivious. As we can see in the resulting e-state (Figure 4.2) r and s believe that the coin lies heads up. Moreover, l and m still do not know the coin position but believe that r and s know it. Finally, being c oblivious, she did not change her beliefs about anything.

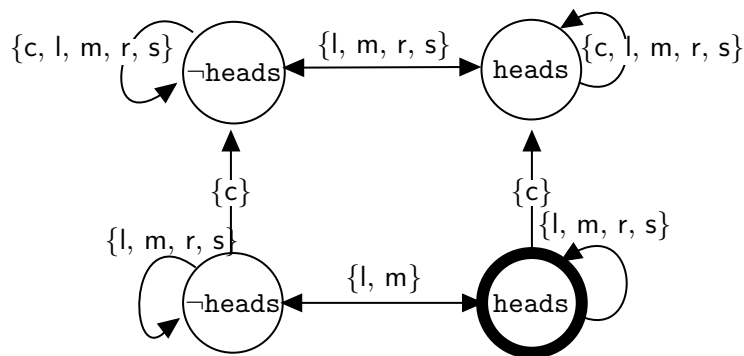


Figure 4.2: The e-state u' obtained after the execution of correct sensing on u .

Example 4.3: Wrong Sensing This example shows how u is updated after the execution of the action instance $\text{peek}\langle\{l, m\}\rangle$. As said in Planning Domain 4.1 both the agents l and m always see the coin lying tails up. Since we are executing a sensing action we are only interested in defining the oblivious, the fully, and the partially observant agents. In particular, for this action instance, we assume l and m to be fully observant, r and s to be partially observant and c to be oblivious. As we can see in the resulting e-state (Figure 4.3) l and m believe that the coin lies tails up. Moreover, r and s still do not know the coin position but believe that l and m know it. Finally, being c oblivious, she did not change her beliefs about anything.

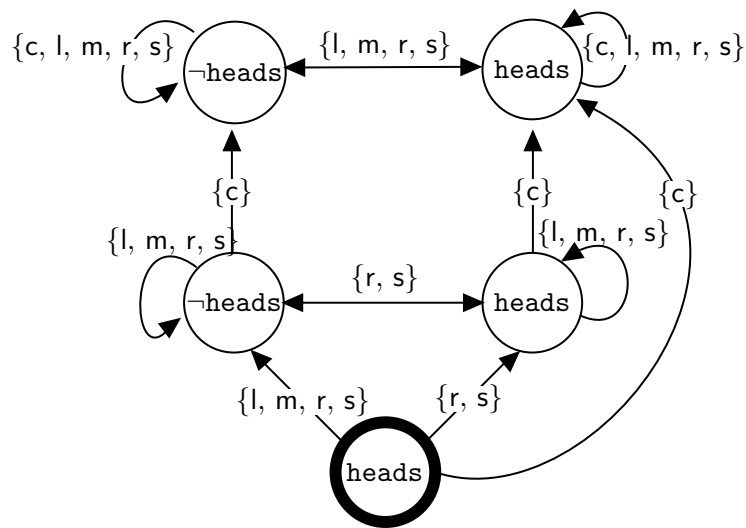


Figure 4.3: The e-state u' obtained after the execution of wrong sensing on u .

Example 4.4: Trust & Mistrust This example shows how u is updated after the execution of the action instance $\text{announce}(c)$ where c announces heads. In particular, for this action instance, we assume:

- c to be the *executor*;
- l to be *trustful*;
- m to be *mistrustful*;
- r to be *impassive*; and
- s to be *doubtful*;

As we can see in the resulting e-state (Figure 4.4) l and m believe that the coin lies heads and tails up, respectively. Moreover, l and m believe that c shares their beliefs on the coin position. Finally, agents r and s , still do not know the coin position but believe that c , l , and m know it.

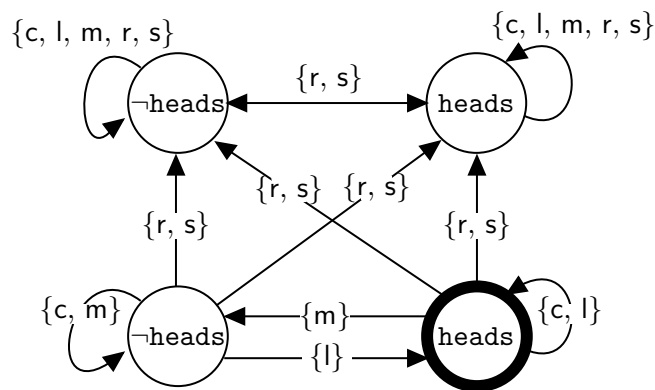


Figure 4.4: The e-state u' obtained after the announcement of heads in u with trustful & mistrustful listeners.

Example 4.5: (Mis)Trust & Stubbornness This example shows how u is updated after the execution of the action instance $\text{announce}(c)$ where c announces **heads**. Differently from the previous example, the agents' attitudes are as follows:

- c to be the *executor*;
- l to be *trustful*;
- m to be *mistrustful*;
- r to be *doubtful*; and
- s to be *stubborn*;

As we can see in the resulting e-state (Figure 4.5) agents c and l believe that the coin lies heads up while m thinks that it lies tails up. Even if s did not change her beliefs on the coin position she knows what c , l believe that the coin is heads up while m think that it is tails up. Agent r , instead still does not know the coin position but believes that c , l , and m know it. We will use a dotted square to indicate that the edges that reach such a square, transitively reach all the worlds contained.

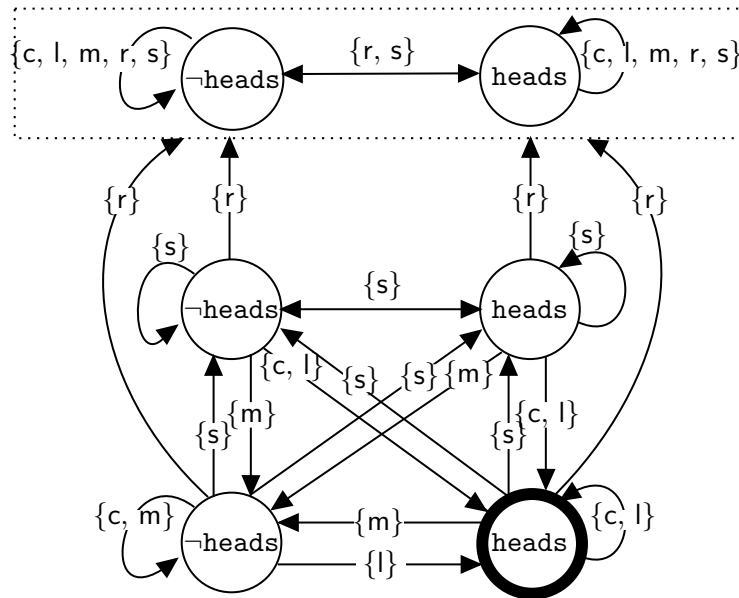


Figure 4.5: The e-state u' obtained after the announcement of **heads** in u with (mis)trustful & stubborn listeners.

Example 4.6: Lie This example shows how u is updated after the execution of the action instance $\text{announce}\langle c \rangle$ where c announces $\neg\text{heads}$. Let us note that this announcement, since it is performed by c that believes heads , is a lie. For this action instance, we assume:

- c to be the *executor*;
- l to be *trustful*;
- m to be *mistrustful*;
- r to be *doubtful*; and
- s to be *oblivious*;

As we can see in the resulting e-state (Figure 4.6) agent l believed to the lie and now has a wrong belief about the coin position. On the other hand and m did not believe the announcer and, therefore, now correctly think that the coin lies heads up. Being the executor, c knew that she was lying and, therefore, still believes that the coin is heads up. Moreover, l and m believe that c shares their beliefs on the coin position. Agents r , still does not know the coin position but believe that c , l , and m know it. Finally, being s oblivious, she did not change her beliefs about anything.

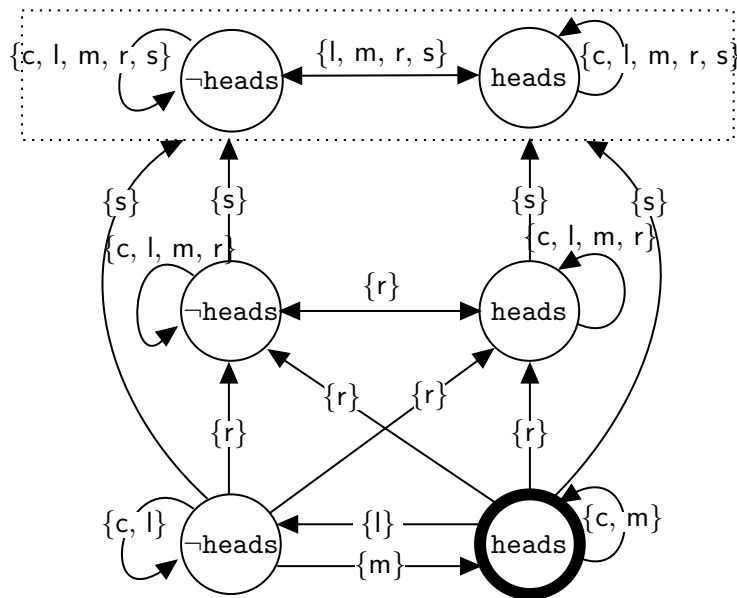


Figure 4.6: The e-state u' obtained after the execution of a lie (*i.e.*, c announce $\neg\text{heads}$) in u with trustful & mistrustful listeners.

4.2.2 Desired Properties

Following the usual schema, we list some properties concerning the new actions that consider attitudes. Complete proofs are available in Appendix A.4.

Proposition 4.1: Epistemic Actions Properties

Let $\mathbf{a}\langle j \rangle$ be an epistemic action instance such that j **announces** ℓ (where ℓ is either \mathbf{f} or $\neg\mathbf{f}$). Let \mathbf{u} be an e-state and let \mathbf{u}' be its updated version, i.e., $\Phi(\mathbf{a}, \mathbf{u}) = \mathbf{u}'$, then it holds that:

- (1) $\mathbf{u}' \models \mathbf{C}_{\mathbf{F}_a}(\mathbf{C}_{\mathbf{T}_a}(\ell \wedge \mathbf{B}_j(\ell)))$;
- (2) $\mathbf{u}' \models \mathbf{C}_{\mathbf{F}_a}(\mathbf{C}_{\mathbf{M}_a}(\neg\ell \wedge \mathbf{B}_j(\neg\ell)))$;
- (3) $\forall i \in (\mathbf{S}_a \cup \{j\})$, $\mathbf{u}' \models \varphi$ if $\mathbf{u} \models \varphi$ with $\varphi \in \{\mathbf{B}_i(\ell); \mathbf{B}_i(\neg\ell); (\neg\mathbf{B}_i(\ell) \wedge \neg\mathbf{B}_i(\neg\ell))\}$;
- (4) $\forall i \in \mathbf{F}_a$, $\mathbf{u}' \models \mathbf{C}_{\mathbf{P}_a}(\mathbf{B}_i(\ell) \vee \mathbf{B}_i(\neg\ell))$;
- (5) $\forall i \in \mathbf{D}_a$, $\mathbf{u}' \models \mathbf{C}_{\mathbf{F}_a \cup \mathbf{P}_a}(\neg\mathbf{B}_i(\ell) \wedge \neg\mathbf{B}_i(\neg\ell))$;
- (6) for every pair of agents $i \in D(\mathcal{AG})$ and $\mathbf{o} \in \mathbf{O}_a$, and a belief formula φ , $\mathbf{u}' \models \mathbf{B}_i(\mathbf{B}_o(\varphi))$ if $\mathbf{u} \models \mathbf{B}_i(\mathbf{B}_o(\varphi))$.

The features presented in Proposition 4.1 capture fundamental aspects of the updated e-state after the execution of an announcement. Intuitively, they model the following properties:

- (1) Fully observant agents think that trustful agents believe that the announced property holds *and* that the announcer believes such property;
- (2) Fully observant agents think that mistrustful agents believe that the announced property does not hold *and* that the announcer does not believe such property;
- (3) Stubborn agents and the announcer do not modify their beliefs about the announced property.
- (4) Partially observant agents believe that fully observant agents (including the announcer) are certain of the value of the announced property;

- (5) Non-oblivious agents believe that doubtful agents are uncertain on the truth value of the announced property;
- (6) Every agent (even oblivious agents) knows that oblivious agents do not change their beliefs.

4.3 Related Work

The enriched semantics of $m\mathcal{A}^p$ has been implemented in the C++ solver EFP (presented in chapter 5) that is now able to tackle families of problems that consider complex aspects such as doxastic reasoning, lying agents, faulty perception, etc.

To the best of our knowledge, in the literature, only one other solver, RP-MEP [Muise et al., 2015], can tackle such domains. This solver firstly encodes a MEP problem into a classical planning problem and then handles the solving phase with a “classical” planner. The key difference between EFP and RP-MEP is that while RP-MEP grounds the agents’ beliefs and reasons on them as if they were “static facts”, EFP builds and interprets e-states, and it updates them using a full-fledged epistemic transition function. For this reason, the latter constitutes a more comprehensive framework. In fact, given the effects of an action instance (a single literal/conjunction of literals), the transition function of $m\mathcal{A}^p$ *propagates* the effects and updates the nested beliefs of agents automatically. Conversely, RP-MEP needs the *propagated* effects to be explicit. Nonetheless, the “implicit beliefs update” of EFP makes this approach less performing with respect to RP-MEP. The latter, in fact, with a little extra effort on the input description, is able to solve the same domains as EFP, outperforming it whenever the depth of the formulae is set to a reasonable number.

Other theoretical approaches explore the idea of trust between agents [Castelfranchi and Falcone, 1998, Herzig et al., 2010, Rodenhäuser, 2014]. For example, following Castelfranchi and Falcone [1998], Herzig et al. devised a logic to capture the “*truster’s belief about certain relevant properties of the trustee with respect to a given goal*”. While the ideas of Castelfranchi and Falcone are elegantly captures

by this logic, Herzig et al. do not actively use the notion of trust to modify the outcome of an action's execution with respect to an agent's perspective, that is what we are trying to accomplish with our idea of attitudes. Conversely, Rodenhäuser [2014] proposes a theoretical framework where agents make use of the reliability of the source (using the so-called *dynamic attitudes*) to correctly update their beliefs. While our idea of attitudes stems from such work, we only introduced attitudes that are intuitively derived from real-world scenarios without considering more complex ones. In the future, we plan to expand our formalization and the planner with the attitudes presented by Rodenhäuser [2014] along with the idea of "*plausibility*".

Belief revision [Baltag and Smets, 2016] in presence of inconsistent/false beliefs has been explored by Baral et al. and Herzig et al.. These works focus on the introduction of a theoretical framework for resolving inconsistencies. Hence, we only compare their approaches with our formalization. Baral et al. [2015] mainly focus on *false beliefs*, dividing their approach into two steps. First, they remove the incorrect beliefs of fully observant agents from the current e-state; and next, they apply the action on the revised state. Their solution correctly accounts for false beliefs, but it is not sufficient to resolve inconsistent beliefs. On the other hand, Herzig et al. [2005] propose a multi-agent extension of AGM-style belief revision [Gärdenfors and Makinson, 1988]. Once a new property is acquired with a sensing action, the agents update their beliefs according to their view of the observation. The revision procedure makes use of a preference-based revision operator. While revising the agents' beliefs could be a viable solution we believe that having to decouple the belief revision from the e-state update for each action execution would generate an excessive overhead in the solving process.

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	103
5.1.1	Imperative and Declarative Programming	104
5.2	EFP: an Epistemic Forward Planner	106
5.2.1	The Overall Architecture	107
5.2.2	EFP 2.0	108
5.2.3	Experimental Evaluation	110
5.2.4	Optimizations and Alternative Search Strategies	120
5.3	PLATO: an Epistemic Planner in ASP	137
5.3.1	Modeling MEP using ASP	137
5.3.2	Experimental Evaluation	145
5.3.3	Correctness of PLATO	147

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:

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

where $0 \leq m \leq n$ and each element a_i , with $0 \leq i \leq n$, is an *atom* of the form $p(t_1, \dots, t_k)$, p is a predicate symbol of arity k and t_1, \dots, 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 $B^+(r) = \{a_1, \dots, a_m\}$ and $B^-(r) = \{a_{m+1}, \dots, a_n\}$ the positive and negative parts of its *body*, respectively; we denote the body with $B(r) = \{a_1, \dots, \text{not } a_n\}$. A rule is called a *fact* whenever $B(r) = \emptyset$; a rule is a *constraint* when its head is empty ($h(r) = \text{false}$); 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 ($\text{false} \notin M$) 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 $sp(p_1, \dots, p_k)$` , 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 $m\mathcal{A}^p$. 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

- 1 Compute the initial e-state given s_0 : (\mathcal{M}_i, W_i)
- 2 Initialize a priority queue $q = [(\{\mathcal{M}_i, W_i\}, [])]$
- 3 **while** q is not empty **do**
- 4 $(\Omega, plan) = dequeue(q)$
- 5 **if** $(\mathcal{M}, W_d) \models \phi_g$ for every $(\mathcal{M}, W_d) \in \Omega$ **then**
- 6 **return** $plan$
- 7 **end**
- 8 **for** action a executable in every (\mathcal{M}, W_d) in Ω **do**
- 9 Compute $\Omega' = \bigcup_{(\mathcal{M}, W_d) \in \Omega} \Phi(a, (\mathcal{M}, W_d))$
- 10 Compute heuristics and insert $(\Omega', plan \circ a)$ into q
- 11 **end**
- 12 **end**
- 13 **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 \mathbf{f} is not considered by φ_{ini} , EFP 2.0 assumes that is common knowledge between all the agents that \mathbf{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-foundedness, 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;
- T0 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).

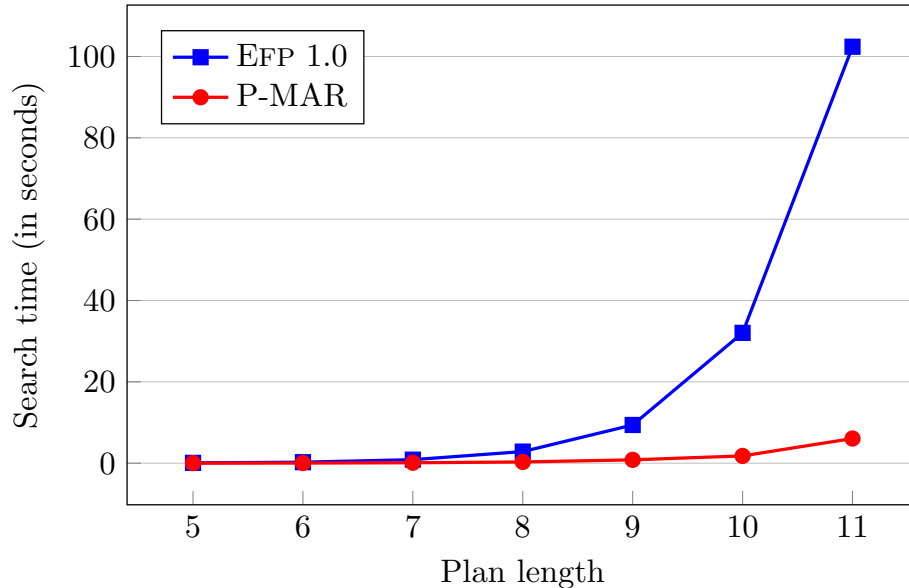


Figure 5.1: Comparison between EFP 1.0 and P-MAR on SC instances with $k = 11$ rooms and $|\mathcal{AG}| = 9$.

(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											
$ \mathcal{AG} $	$ \mathcal{F} $	$ \mathcal{A} $	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 $ \mathcal{AG} = 3$, $ \mathcal{F} = 8$, $ \mathcal{A} = 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	T0	T0	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 $m\mathcal{A}^p$ 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-MAR after 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 $ \mathcal{AG} = 3$, $ \mathcal{F} = 8$, $ \mathcal{A} = 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 $m\mathcal{A}^p$ Update As mentioned above, EFP 2.0, besides implementing the language $m\mathcal{A}^p$, 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 $m\mathcal{A}^p$ 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 $ \mathcal{AG} = 4$, $ \mathcal{F} = 16$, $ \mathcal{A} = 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 $m\mathcal{A}^p$. 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-states¹, 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.

¹Figures 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	%
4	0.119	0.087	26.9	3	0.166	0.087	47.9
5	0.864	0.623	27.9	4	0.846	0.459	45.7
6	3.000	1.830	27.5	5	14.980	7.950	46.8
7	23.453	16.816	25.7	6	47.330	25.490	46.1
				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 search², 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.

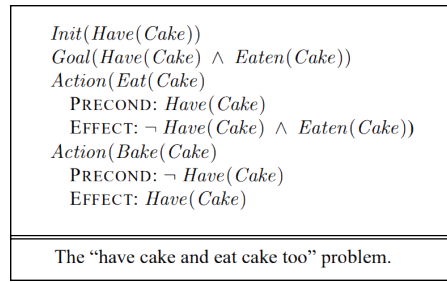
That is why, as the final contribution to EFP 2.1, we decided to focus on formalizing some domain-independent heuristics for MEP. First of all, we implemented a module, called `heuristics_manager`, that allows the planner to make use of heuristics as black boxes. This allows any interested researcher 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.

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

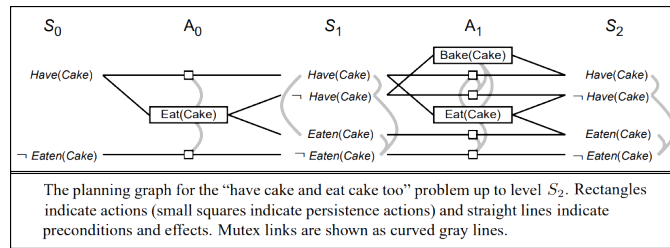
The first is a very simple heuristic that simply associates an higher evaluation to e-states that satisfy more sub-goals. To better improve this heuristic we also defined functions that allows “to break” complex goals into a conjunction of simpler ones. That is, being each goal simply a belief formula 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:



(a) Problem description.



(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 $S_0, A_0, \dots, S_k, A_k, \dots$ 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 $S_i \models \varphi$ (where \models 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 executable³ action \mathbf{a} and a state level S_i we have that \mathbf{a} could potentially set to true the Boolean value associated with any fluent literal⁴ 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 \mathbf{f} then the update will check if this fluent, its negation, or the belief formulae that have this \mathbf{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 \mathbf{f} True than $Q[\mathbf{f}] = \top$, otherwise, if the action sets \mathbf{f} to be False, we will have $Q[\neg\mathbf{f}] = \top$. A more precise definition of this procedure is illustrated in Algorithm 2.

Algorithm 2: Fluent value updater

```

Input  :  $Q$     //The map (fluent, bool) of  $S_i$ 
            $\mathbf{a}$     //The action executed on  $S_i$ 
Output:  $Q$     //The updated version of the input map
1 //Note that  $\ell$  may also be a negated fluent
2 for fluent  $\ell$  in  $Q.get\_keys()$  do
3   | if  $Q[\ell].get\_value() == \perp$  then
4   |   | //An effect is considered if its condition are True
5   |   | if  $\mathbf{a}.get\_effects().contains(\ell)$  then
6   |   |   |  $Q[\ell].set\_value(\top)$ 
7   |   |   | end
8   |   | end
9 end
10 return  $Q$ 

```

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

⁴Let us remember that each fluent literal and its negation are independent and are considered as separate entries in 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 :  $\mathcal{P}$  //The map  $\langle \text{belief\_formula}, \text{bool} \rangle$  of  $S_i$ 
         a //The epistemic action executed on  $S_i$ 
Output:  $\mathcal{P}$  //The updated map  $\langle \text{belief\_formula}, \text{bool} \rangle$  of  $S_i$ 
1 for belief_formula bf in  $\mathcal{P}.\text{get\_keys}()$  do
2   if  $\mathcal{P}[\text{bf}].\text{get\_value}() == \perp$  then
3     if  $\mathcal{P}[\text{bf}].\text{get\_type}() == \text{ontic}$  then
4       //Call to Algorithm 4
5       bool res = check_bf(bf, a,  $\mathcal{P}$ )
6     else
7       //Call to Algorithm 5
8       bool res = check_bf_epi(bf, a,  $\mathcal{P}$ , 0)
9     end
10  end
11   $\mathcal{P}[\text{bf}].\text{set\_value}(\text{res})$ 
12 end
13 return  $\mathcal{P}$ 

```

Algorithm 4: Check belief formula after ontic execution

```

Input :bf          //The belief formula to verify
          $\mathcal{P}$       //The map  $\langle \text{belief\_formula}, \text{bool} \rangle$  of  $S_i$ 
         a           //The ontic action executed on  $S_i$ 
Output:  $\top$  or  $\perp$  //Depending on the updated value of  $\mathcal{P}[\text{bf}]$ 
1 if  $\mathcal{P}[\text{bf}].\text{get\_value}() == \top$  then
2 |   return  $\top$ 
3 end
4
5 if  $\text{bf}.\text{get\_base\_fluents}().\text{contains\_one}(a.\text{get\_effects}())$  then
6 |   if  $\text{bf}.\text{get\_type}() == \text{fluent\_formula}$  then
7 | |   //For simplicity, assume fluent formulae of one fluent
8 | |    $\mathcal{P}[\text{bf}].\text{set\_value}(\top)$ 
9 | |   return  $\top$ 
10 |  else if  $\text{bf}.\text{get\_type}() == \text{single\_ag\_belief}$  then
11 | |   if  $\text{fully\_obs}.\text{contains}(\text{bf}.\text{get\_agent}())$  then
12 | | |   //Recursive call to this function
13 | | |   return check_bf( $\text{bf}.\text{get\_nested\_bf}()$ , a,  $\mathcal{P}$ )
14 | |   end
15 |  else if  $\text{bf}.\text{get\_type}() == \text{group\_formula}$  then
16 | |   if  $\text{fully\_obs}.\text{contains}(\text{bf}.\text{get\_group\_agents}())$  then
17 | | |   //Recursive call to this function
18 | | |   return check_bf( $\text{bf}.\text{get\_nested\_bf}()$ , a,  $\mathcal{P}$ )
19 | |   end
20 |  else
21 | |   /*Disjunction and conjunction of belief formulae follow
22 | |   the standard semantics of these operators*/
23 end
24 return  $\perp$ 

```

Algorithm 5: Check belief formula after epistemic execution

```

Input :bf      //The belief formula to verify
          $\mathcal{P}$     //The map  $\langle \text{belief\_formula}, \text{bool} \rangle$  of  $S_i$ 
         a      //The ontic action executed on  $S_i$ 
         x      //Label used to differentiate scenarios
Output:  $\top$  or  $\perp$  //Depending on the updated value of  $\mathcal{P}[\text{bf}]$ 
1 if  $\mathcal{P}[\text{bf}].\text{get\_value}() == \top$  then
2 |   return  $\top$ 
3 end
4
5 //We check also for negated effects for the Partial observers
6 if  $\text{bf}.\text{get\_base\_fluent}().\text{contains\_one\_or\_negated}(a.\text{get\_effects}())$  then
7 |   if  $\text{bf}.\text{get\_type}() == \text{fluent\_formula}$  then
8 | |   if  $a.\text{get\_effects}().\text{contains}(\text{bf}.\text{get\_base\_fluent}())$  and  $x == 0$  then
9 | | |   return  $\top$ 
10 | |   else if  $x == 1$  then
11 | | |   return  $\top$ 
12 | |   else
13 | | |   return  $\perp$ 
14 | |   end
15 |   else if  $\text{bf}.\text{get\_type}() == \text{single\_ag\_belief}$  then
16 | |   if  $\text{fully\_obs}.\text{contains}(\text{bf}.\text{get\_agent}())$  then
17 | | |   if  $x == 2$  then
18 | | | |    $x = 1$ 
19 | | |   end
20 | | |   return check_bf( $\text{bf}.\text{get\_nested\_bf}()$ , a,  $\mathcal{P}$ , x)
21 | |   else if  $\text{partially\_obs}.\text{contains}(\text{bf}.\text{get\_agent}())$  then
22 | | |   return check_bf( $\text{bf}.\text{get\_nested\_bf}()$ , a,  $\mathcal{P}$ , 2)
23 |   else if  $\text{bf}.\text{get\_type}() == \text{group\_formula}$  then
24 | |   if  $\text{fully\_obs}.\text{contains}(\text{bf}.\text{get\_group\_agents}())$  then
25 | | |   return check_bf( $\text{bf}.\text{get\_nested\_bf}()$ , a,  $\mathcal{P}$ , 0)
26 | |   else if  $\text{partially\_obs}.\text{contains\_one}(\text{bf}.\text{get\_group\_agents}())$  then
27 | | |   return check_bf( $\text{bf}.\text{get\_nested\_bf}()$ , a,  $\mathcal{P}$ , 2)
28 |   else
29 | |   /*Disjunction and conjunction of belief formulae follow
30 | |   the standard semantics of these operators*/
31 end
32 return  $\perp$ 

```

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 point⁵:
 - 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].

⁵A 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 $m\mathcal{A}^p$. 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 $\text{pos}_w(T, R, P)$. 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 $v \in u(i)$ 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 $u(F) = 1$. 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, $\mathbf{f} \in D(\mathcal{F})$ be a fluent, $i \in D(\mathcal{AG})$ be an agent, and let us use \mathcal{AG} instead of $D(\mathcal{AG})$ for the sake of readability. Consider a $m\mathcal{A}^p$ statement of the form $[\mathbf{initially} \varphi] \in D$; we have five cases:

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

(ii) $\varphi \equiv \mathbf{C}_{\mathcal{AG}}(\mathbf{f}/\neg\mathbf{f})$: \mathbf{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) $\varphi \equiv \mathbf{C}_{AG}(\mathbf{B}_i(\psi) \vee \mathbf{B}_i(\neg\psi))$: there can be no two possibilities \mathbf{u} and \mathbf{v} such that $\mathbf{v} \in \mathbf{u}(i)$ 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) $\varphi \equiv \mathbf{C}_{AG}(\neg\mathbf{B}_i(\psi) \wedge \neg\mathbf{B}_i(\neg\psi))$: 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 \mathbf{u} , there must exist $\mathbf{v} \in \mathbf{u}(i)$ such that $\mathbf{u} \models \psi$ and $\mathbf{v} \not\models \psi$ (or $\mathbf{u} \not\models \psi$ and $\mathbf{v} \models \psi$).

Formulae of types (i)–(iii) are used to build the fluent sets of the possible worlds within the initial state. A fluent \mathbf{f} is *initially known* if there exists a statement $[\mathbf{initially} \ \mathbf{C}_{AG}(\mathbf{f})]$ or $[\mathbf{initially} \ \mathbf{C}_{AG}(\neg\mathbf{f})]$. In the former case, all agents will believe that \mathbf{f} is true, whereas in the latter that \mathbf{f} is false. If there are no such statements for \mathbf{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 $\mathbf{P} \in \{1, \dots, 2^{uk}\}$, one for each possible truth combination of such fluents. For each initial possibility \mathbf{P} and each initially known fluent \mathbf{F} , we create an atom $\mathbf{holds}(0, 0, \mathbf{P}, \mathbf{F})$ ⁶, since it is common belief between all agents that \mathbf{F} is true (we deal with negated fluents similarly). Moreover, through the atoms \mathbf{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 $\mathbf{pos_w}(0, 0, \mathbf{P})$ for every *good* initial possibility \mathbf{P} . The initial pointed possibility is specified by $\mathbf{pointed}(0, 0, \mathbf{PP})$, where \mathbf{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 $\mathbf{P1}$ and $\mathbf{P2}$ be two *good* initial possibilities; the atom $\mathbf{believes}(0, 0, \mathbf{P1}, 0, 0, \mathbf{P2}, \mathbf{AG})$ holds if there are no initial type (iv) formulae ψ such that $\mathbf{P1}$ and $\mathbf{P2}$ do *not* agree

⁶Let 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, $\forall \mathbf{u} \in \mathcal{G}, \forall i \in \mathcal{AG}$ we have that $\mathbf{u}(i) = \mathcal{G}$. 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 $T > 0$. $m\mathcal{A}^p$ 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 `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 `(T2, R2, P2)`. Without loss of generality, we represent an action instance by a unique action (using fresh actions names). Let us describe how we have modeled these actions in ASP.

Ontic actions Let `ACT` be an ontic action executed at time T and let `u = (T-1, RP, PP)` be the pointed possibility at time $T-1$. Intuitively, when an ontic action is executed, the resulting possibility `u'` is calculated by applying the action effects on `u` and also on the possibilities `w $\in u(i)$` , 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:

$$\begin{aligned} \text{pos_w } (T, R2 + MR + 1, P2) :- \\ \text{pointed}(T-1, RP, PP), \text{ pos_w}(T2, R2, P2), T2 < T, \\ \text{reaches}(T-1, RP, PP, T2, R2, P2, AGS), \text{ subset}(AGS, F_{ACT}). \end{aligned}$$

where `MR` is the maximum value of the parameter *repetition* among all the possibilities at time $T-1$ and `FACT` represents the set of fully observant agents of `ACT`. Hence, if

$(T, R2, P2)$ 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 $\text{holds}(P2', F)$ if $\text{holds}(P2, F)$, for each fluent F (inertia).

Let **AG** be a partially observant agent. If $\text{believes}(P1, P2, \text{AG})$ holds, then we will impose $\text{believes}(P1', P2', \text{AG})$, 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 $MF_i = \#max \{ F : holds(P_i, F), not\ holds(P_j, F) \}$, with $i \neq j$. Then we impose that if $P1 < P2$, then it must also hold that $MF1 < MF2$. Since it could be the case that there exist finitely many initial states, by implementing this constraint we are able to generate a unique initial state while discarding the (possible) other equivalent ones.

Multi-shot encoding

Following the approach of Gebser et al. [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 ($t = 0$), alongside with the instance encoding; (2) `step(t)`: it deals with the plan generation ($t > 0$) and with the application of the transition function; and (3) `check(t)`: it verifies whether the goal is reached at time $t \geq 0$.

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 $t + 1$ (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 $\text{step}(t)$ ($t > 0$) 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 ($T0$) 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	T0	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	T0	28.02	.54	.10
5	T0	7.68	1.14	.16	T0	16.13	4.89	.60
6	1222.67	10.83	4.42	0.64	T0	14.84	12.66	1.71
7	T0	30.08	16.06	2.61	T0	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 $\Gamma : D(\mathcal{AI}) \times D(\mathcal{S}) \rightarrow D(\mathcal{S}) \cup \{\emptyset\}$ (where $D(\mathcal{AI})$ and $D(\mathcal{S})$ are defined as in Definition 1.15). Finally, we express the entailment with respect to $m\mathcal{A}^p$ and PLATO with \models_{Φ} and \models_{Γ} , 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 $u \in D(\mathcal{S})$ we have that $\forall \psi \in D(\mathcal{BF})$ $u \models_{\Phi} \psi$ iff $u \models_{\Gamma} \psi$.

Proposition 5.2: PLATO Initial State Construction Correctness

Given two possibilities $u, v \in D(\mathcal{S})$ such that u is the initial state in $m\mathcal{A}^p$ and v is the initial state in PLATO then $\forall \psi \in D(\mathcal{BF})$ $u \models_{\Phi} \psi$ iff $v \models_{\Gamma} \psi$.

Proposition 5.3: PLATO Transition Function Correctness

Given an action instance $\mathbf{a} \in D(\mathcal{AI})$ and two possibilities $\mathbf{u}, \mathbf{v} \in D(\mathcal{S})$ such that $\forall \psi \in D(\mathcal{BF}) \mathbf{u} \models_{\Phi} \psi$ iff $\mathbf{v} \models_{\Gamma} \psi$ then $\forall \psi \in D(\mathcal{BF}) \Phi(\mathbf{a}, \mathbf{u}) \models_{\Phi} \psi$ iff $\Gamma(\mathbf{a}, \mathbf{v}) \models_{\Gamma} \psi$.

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 $m\mathcal{A}^p$. 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	149
6.1.1	Theories of Human Decision Making	151
6.1.2	AI Thinking, Fast and Slow	153
6.2	MEP System-1 and System-2	153
6.2.1	Meta-cognition	156
6.3	A Fast and Slow Epistemic Architecture	159
6.3.1	E-PDDL: Standardized MEP Problems Language	159
6.3.2	The Overall Architecture	165

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 “**Systems**”. 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 **Systems** 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*.

¹We 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.

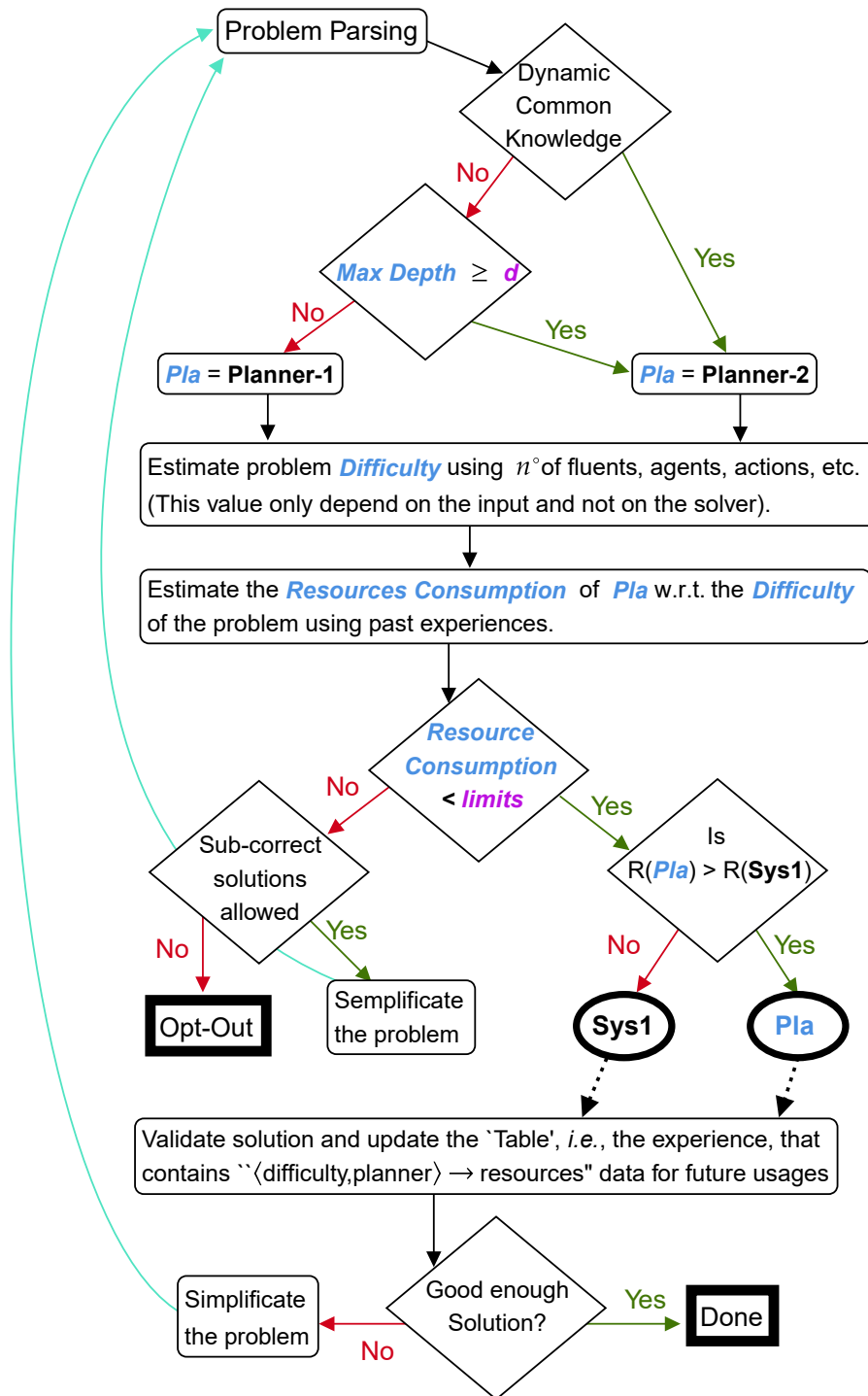


Figure 6.1: The schema of MC-2.

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 $i \in \mathcal{AG}$. This operator captures the modal operator \mathbf{B}_i . For example, in Line 10 of Listing 6.1, the formula “[*i*](has_key ?i)” reads “agent *i* knows has_key_i” where $i \in \mathcal{AG}$ and the fluent `has_key_i` encodes the fact that *i* has the key. When the operator is of the form “[α]” 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 $m\mathcal{A}^p$ 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 *observers*. 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 $m\mathcal{A}^p$, 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 $\mathbf{B}_i(\varphi)$ has depth 1 while $\mathbf{B}_i(\mathbf{C}_{i,j}(\varphi))$ 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 $\ell \in \{0, \dots, d\}$ and d is the value assigned to the field `:depth` in the problem-*instance* (in Listings 6.2 $d = 2$). 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 $\ell = d$.

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 $\ell \leq d - 2$, 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-MEP². 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 implementation³ 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

²We 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.

³Available 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.

⁴Available at <https://github.com/QuMuLab/pdkb-planning>.

[...] ἔοικα γοῦν τούτου γε μικρῶ τι ἀπὸ τούτω
σοφώτερος εἶναι, ὅτι ἂ μὴ οἶδα οὐδὲ οἶμαι 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 $m\mathcal{A}^p$, an action language for MEP based on possibilities—a non-well-founded data structure. $m\mathcal{A}^p$ 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 $m\mathcal{A}^p$ —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



Propositions Proofs

Contents

A.1 Preliminary Definitions	171
A.2 Proofs of Propositions 2.3 to 2.5	174
A.3 Proofs of Propositions 3.1 and 3.2	179
A.3.1 Updated States Size Finiteness	179
A.3.2 Proofs	180
A.4 Proof of Proposition 4.1	186
A.5 Proofs of Propositions 5.1 to 5.3	192
A.5.1 Abbreviations	192
A.5.2 PLATO Entailment Correctness	192
A.5.3 PLATO Initial State Construction Correctness	194
A.5.4 PLATO Transition Function Correctness	196

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 $\mathfrak{p} \in \mathcal{S}$ 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 $\mathcal{B}_{\mathcal{AG}}^{\mathfrak{p}}$ captures *all the reachable possibilities for \mathcal{AG} given a starting possibility \mathfrak{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.*, $\mathcal{AG} = \{i\}$. We, therefore, use \mathcal{B}_i^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}_i^p is procedural and it is done by applying the operator $(\mathcal{B}_i^p)^k$, with $k \in \mathbb{N}$, until a *fixed point* is found. The operator $(\mathcal{B}_i^p)^k$ is defined as follows:

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

Finally, we can define $\mathcal{B}_i^p = \bigcup_{k \geq 1} (\mathcal{B}_i^p)^k$. It is easy to see that this is equivalent to the set of possibilities reached by the operator \mathbf{B}_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 $(\mathcal{B}_{\mathcal{AG}}^S)^k$ is reached in a finite number of iterations. This is because:

- $(\mathcal{B}_{\mathcal{AG}}^S)^k$ is monotonic; namely $(\mathcal{B}_{\mathcal{AG}}^S)^k \subseteq (\mathcal{B}_{\mathcal{AG}}^S)^{k+1}$ 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 $\mathcal{B}_{\mathcal{AG}}^p$. This represents the *common belief* of \mathcal{AG} ($\mathbf{C}_{\mathcal{AG}}$) starting from p . As before we introduce the operator $(\mathcal{B}_{\mathcal{AG}}^p)^k$ of which the fixed point will result in $\mathcal{B}_{\mathcal{AG}}^p$.

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

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 \mathbf{p} is:

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

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

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

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

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

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

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 $\mathbf{q} \in \mathbf{p}(i)$ then $\mathbf{q} \in \mathbf{q}(i)$. In terms of our sequence, this translates into *if a possibility $\mathbf{q} \in (\mathcal{B}_i^{\mathbf{p}})^{k-1}$ then $\mathbf{q} \in (\mathcal{B}_i^{\mathbf{p}})^k$* .

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

Lemma 1.2: States Size Finiteness

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

Proof of Lemma A.2 Following the definition of the transition function of $m\mathcal{A}^p$ (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 \mathbf{p} such that $|\mathcal{B}_{\mathcal{AG}}^{\mathbf{p}}| = n$ (where \mathcal{AG} is the set of all the agents) the cardinality of the set $\mathcal{B}_{\mathcal{AG}}^{\mathbf{p}'}$ will be, at most, equal to $2n$. That is because:

- when an *ontic* action is executed each possibility $\in |\mathcal{B}_{\mathcal{AG}}^{\mathbf{p}}|$ 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 $\mathcal{B}_{\mathcal{AG}}^{\mathbf{p}'}$ 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. \square

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

- As said previously $\mathcal{B}_x^{u'}$ represents \mathbf{B}_x starting from u' .
 - It is easy to see that if every element in $\mathcal{B}_x^{u'}$ entails \mathbf{f} , then $u' \models \mathbf{B}_x(\mathbf{f})$.
- (2) As in the previous item, we assume action \mathbf{a} to be executable in \mathbf{u} meaning that:
- If $\mathbf{u} \models \mathbf{B}_y(\varphi)$ we have that every $\mathbf{p} \in \mathcal{B}_y^{\mathbf{u}}$ entails φ .
 - From Definition 2.12 when $\mathbf{y} \in \mathbf{O}_a$ for each possibility $\mathbf{p} \in \mathcal{B}_y^{\mathbf{u}}$ $\mathbf{p}(\mathbf{y}) = \mathbf{p}'(\mathbf{y})$ it is easy to see that $\mathcal{B}_y^{\mathbf{u}} \equiv \mathcal{B}_y^{u'}$.
 - 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 $\forall \varphi \in D \mathbf{u} \models \mathbf{B}_y(\varphi)$ iff $u' \models \mathbf{B}_y(\varphi)$.
- (3) Again we assume the executability of the action \mathbf{a} and we consider $\mathbf{x} \in \mathbf{F}_a$ and $\mathbf{y} \in \mathbf{O}_a$:
- Being $\mathbf{y} \in \mathbf{O}_a$, from Definition 2.12, we know that $\mathbf{p}(\mathbf{y}) = \mathbf{p}'(\mathbf{y})$ such that $\mathbf{p} \in \mathcal{B}_x^{\mathbf{u}}$ and \mathbf{p}' is its updated version $\in \mathcal{B}_x^{u'}$.
 - This means that for every element in $\mathcal{B}_x^{\mathbf{u}}$ we have an updated version that has the same reachability function for the agent \mathbf{y} .
 - Then it is easy to see that $\mathcal{B}_{x,y}^{\mathbf{u}} \equiv \mathcal{B}_{x,y}^{u'}$ 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 $\mathbf{u} \models \mathbf{B}_x(\mathbf{B}_y(\varphi))$ then $u' \models \mathbf{B}_x(\mathbf{B}_y(\varphi))$

□

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

(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 $\mathcal{B}_{\mathbf{P}_a}^{\mathbf{u}}$. We also remind that this set is equal to $\mathbf{C}_{\mathbf{P}_a}$ in \mathbf{u} .

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

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

Where $i \in \mathbf{P}_a$

- It is easy to identify two disjoint subsets $\mathcal{B}_{\mathbf{P}_a}^1$ and $\mathcal{B}_{\mathbf{P}_a}^2$ of $\mathcal{B}_{\mathbf{P}_a}^{u'}$ that contains only possibility such that:
 - $\mathcal{B}_{\mathbf{P}_a}^1 \models e(\mathbf{a}, \mathbf{u})$;
 - $\mathcal{B}_{\mathbf{P}_a}^2 \not\models e(\mathbf{a}, \mathbf{u})$;
 - $(\mathcal{B}_{\mathbf{P}_a}^1 \cup \mathcal{B}_{\mathbf{P}_a}^2) \equiv \mathcal{B}_{\mathbf{P}_a}^{u'}$; and
 - $(\mathcal{B}_{\mathbf{P}_a}^1 \cap \mathcal{B}_{\mathbf{P}_a}^2) \equiv \emptyset$.
- From these two sets we can now construct the sets $\mathcal{B}_{\mathbf{P}_a, \mathbf{F}_a}^1$ and $\mathcal{B}_{\mathbf{P}_a, \mathbf{F}_a}^2$ that are simply the set of possibilities reachable from the *fully observant* agents starting from $\mathcal{B}_{\mathbf{P}_a}^1$ and $\mathcal{B}_{\mathbf{P}_a}^2$, respectively.
- Given that the set $\mathcal{B}_{\mathbf{P}_a, \mathbf{F}_a}^1$ 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 $\forall \mathbf{p} \in \mathcal{B}_{\mathbf{P}_a, \mathbf{F}_a}^1, \mathbf{p} \models \mathbf{f}$.
- This implies that $\mathcal{B}_{\mathbf{P}_a, \mathbf{F}_a}^1$ reaches only possibilities where the interpretation of \mathbf{f} is true and similarly in $\mathcal{B}_{\mathbf{P}_a, \mathbf{F}_a}^2$ only possibilities where the interpretation of \mathbf{f} is false.
- This means that $\mathcal{B}_{\mathbf{P}_a, \mathbf{F}_a}^1 \models \mathbf{f}$ and $\mathcal{B}_{\mathbf{P}_a, \mathbf{F}_a}^2 \models \neg \mathbf{f}$.
- It is easy to see then that $\mathcal{B}_{\mathbf{P}_a}^1 \models \mathbf{C}_{\mathbf{F}_a} \mathbf{f}$ being $\mathcal{B}_{\mathbf{P}_a, \mathbf{F}_a}^1 = \{\mathbf{p} \mid \mathbf{p} \in \bigcup_{\mathbf{q} \in \mathcal{B}_{\mathbf{P}_a}^1} \mathbf{q}(\mathbf{F}_a)\}$ (and similarly $\mathcal{B}_{\mathbf{P}_a}^2 \models \mathbf{C}_{\mathbf{F}_a} \neg \mathbf{f}$).
- Finally being $\mathcal{B}_{\mathbf{P}_a}^{u'} = \mathcal{B}_{\mathbf{P}_a}^1 \cup \mathcal{B}_{\mathbf{P}_a}^2$ we can conclude that $\mathcal{B}_{\mathbf{P}_a}^{u'} \models \mathbf{C}_{\mathbf{F}_a} \mathbf{f} \vee \mathbf{C}_{\mathbf{F}_a} \neg \mathbf{f}$ and therefore $u' \models \mathbf{C}_{\mathbf{P}_a}(\mathbf{C}_{\mathbf{F}_a} \mathbf{f} \vee \mathbf{C}_{\mathbf{F}_a} \neg \mathbf{f})$.

(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 $\mathcal{B}_{\mathbf{F}_a, \mathbf{P}_a}^u$ corresponds with the set of possibilities identified by $\mathbf{C}_{\mathbf{F}_a} \mathbf{C}_{\mathbf{P}_a}$ and it is also equal to $\{\mathbf{p} \mid (\exists \mathbf{q} \in \mathcal{B}_{\mathbf{P}_a}^u)(\mathbf{p} \in \bigcup_{i \in \mathbf{F}_a} \mathbf{q}(i))\}$.
- Now to calculate $\mathcal{B}_{\mathbf{F}_a}^{u'}$ we apply Definition 2.12 to every element of $\mathcal{B}_{\mathbf{F}_a}^u$. This means that $\mathcal{B}_{\mathbf{F}_a}^{u'} = \{\mathbf{p}' \mid (\exists \mathbf{p} \in \mathcal{B}_{\mathbf{F}_a}^u)(\mathbf{p}' = \Phi(\mathbf{a}, \mathbf{p}))\}$.
- We then want to calculate the set $\{\mathbf{p}' \mid (\exists \mathbf{q}' \in \mathcal{B}_{\mathbf{F}_a}^{u'})(\mathbf{p}' \in \bigcup_{i \in \mathbf{P}_a} \mathbf{q}'(i))\}$.
- 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

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

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

(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. □

^aThe two sets are completely disjoint as one only contains possibilities that entail \mathbf{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 $u \models \phi$. The case when $u \models \neg\phi$ 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 $m\mathcal{A}^p$ (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 \mathbf{p} such that $|\mathcal{B}_{\mathcal{AG}}^{\mathbf{p}}| = n$ (where \mathcal{AG} is the set of all the agents) the cardinality of the set $\mathcal{B}_{\mathcal{AG}}^{\mathbf{p}'}$ will be, at most, equal to $3n$. That is because:

- when an *ontic* action is executed each possibility $\in |\mathcal{B}_{\mathcal{AG}}^{\mathbf{p}}|$ 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 $\mathcal{B}_{\mathcal{AG}}^{\mathbf{p}'}$ 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 $\in |\mathcal{B}_{AG}^p|$ 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 $\mathcal{B}_{AG}^{p'}$ 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. \square

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

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

- In order to compute $\mathcal{B}_{\mathbf{U}_a}^{u'}$, following Definition 3.2, we apply $\Psi(\mathbf{a}, \mathbf{u})$ to every element of $\mathcal{B}_{\mathbf{U}_a}^u$.
- 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 $\mathcal{B}_{\mathbf{U}_a, \mathbf{F}_a}^{u'} = \{\mathbf{p}' \mid \mathbf{p} \in \mathcal{B}_{\mathbf{U}_a}^{u'} \wedge e(\mathbf{a}, \mathbf{p}') = 0\}$.
- We then have that $\forall p' \in \mathcal{B}_{\mathbf{U}_a, \mathbf{F}_a}^{u'} p' \models \phi$ and therefore that $u' \models \mathbf{C}_{\mathbf{U}_a}(\mathbf{C}_{\mathbf{F}_a} \phi)$.

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

- Now to calculate $\mathcal{B}_{\mathbf{P}_a}^{u'}$, following Definition 3.2, we apply $\Upsilon(\mathbf{a}, \mathbf{w}) \cup \Psi(\mathbf{a}, \mathbf{u})$ to every element of $\mathcal{B}_{\mathbf{P}_a}^u$.
- It is easy to identify two disjoint subsets $\mathcal{B}_{\mathbf{P}_a}^\Upsilon$ and $\mathcal{B}_{\mathbf{P}_a}^\Psi$ of $\mathcal{B}_{\mathbf{P}_a}^{u'}$ that contain only possibilities such that:
 - $\mathcal{B}_{\mathbf{P}_a}^\Psi \models \mathbf{u}(\mathbf{a}, \mathbf{u})$;
 - $\mathcal{B}_{\mathbf{P}_a}^\Upsilon \not\models \mathbf{u}(\mathbf{a}, \mathbf{u})$;
 - $(\mathcal{B}_{\mathbf{P}_a}^\Psi \cup \mathcal{B}_{\mathbf{P}_a}^\Upsilon) \equiv \mathcal{B}_{\mathbf{P}_a}^{u'}$; and
 - $(\mathcal{B}_{\mathbf{P}_a}^\Psi \cap \mathcal{B}_{\mathbf{P}_a}^\Upsilon) \equiv \emptyset$.
- From these two sets we can now construct the sets $\mathcal{B}_{\mathbf{P}_a, \mathbf{F}_a}^\Psi$ and $\mathcal{B}_{\mathbf{P}_a, \mathbf{F}_a}^\Upsilon$ that are simply the set of possibilities reachable from the *fully observant* agents starting from $\mathcal{B}_{\mathbf{P}_a}^\Psi$ and $\mathcal{B}_{\mathbf{P}_a}^\Upsilon$, respectively.
- Given that the set $\mathcal{B}_{\mathbf{P}_a, \mathbf{F}_a}^\Psi$ 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 $\forall \mathbf{p} \in \mathcal{B}_{\mathbf{P}_a, \mathbf{F}_a}^\Psi, \mathbf{p} \models \phi$.
- This implies that $\mathcal{B}_{\mathbf{P}_a, \mathbf{F}_a}^\Psi$ reaches only possibilities where the interpretation of ϕ is true and similarly in $\mathcal{B}_{\mathbf{P}_a, \mathbf{F}_a}^\Upsilon$ only possibilities where the interpretation of ϕ is false.
- This means that $\mathcal{B}_{\mathbf{P}_a, \mathbf{F}_a}^\Psi \models \phi$ and $\mathcal{B}_{\mathbf{P}_a, \mathbf{F}_a}^\Upsilon \models \neg\phi$.
- We can then derive that $\mathcal{B}_{\mathbf{P}_a}^{u'} \models \mathbf{C}_{\mathbf{F}_a} \phi$ being $\mathcal{B}_{\mathbf{P}_a, \mathbf{F}_a}^\Psi = \{\mathbf{p} \mid \mathbf{p} \in \bigcup_{\mathbf{q} \in \mathcal{B}_{\mathbf{P}_a}^\Psi} \mathbf{q}(\mathbf{F}_a)\}$ (and similarly $\mathcal{B}_{\mathbf{P}_a}^{u'} \models \mathbf{C}_{\mathbf{F}_a} \neg\phi$).
- Finally, being $\mathcal{B}_{\mathbf{P}_a}^{u'} = \mathcal{B}_{\mathbf{P}_a}^\Psi \cup \mathcal{B}_{\mathbf{P}_a}^\Upsilon$ we can conclude that $\mathcal{B}_{\mathbf{P}_a}^{u'} \models \mathbf{C}_{\mathbf{F}_a} \phi \vee \mathbf{C}_{\mathbf{F}_a} \neg\phi$ and therefore $u' \models \mathbf{C}_{\mathbf{P}_a}(\mathbf{C}_{\mathbf{F}_a} \phi \vee \mathbf{C}_{\mathbf{F}_a} \neg\phi)$.

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

(5) Let us consider $y \in \mathbf{O}_a$.

- If $u \models \mathbf{B}_y(\varphi)$ we have that every $\mathbf{p} \in \mathcal{B}_y^u$ entails φ .
- Given that, from Definition 3.2, when $y \in \mathbf{O}_a$ for each possibility $\mathbf{p} \in \mathcal{B}_y^u$ $\mathbf{p}(y) = \mathbf{p}'(y)$ it is easy to see that $\mathcal{B}_y^u \equiv \mathcal{B}_y^{u'}$.
- 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 $\forall \varphi \in D$ $u \models \mathbf{B}_y(\varphi)$ iff $u' \models \mathbf{B}_y(\varphi)$.

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

- Being $y \in \mathbf{O}_a$, from Definition 3.2, we know that $\mathbf{p}(y) = \mathbf{p}'(y)$ such that $\mathbf{p} \in \mathcal{B}_x^u$ and \mathbf{p}' is its updated version $\in \mathcal{B}_x^{u'}$.
 - This means that for every element in \mathcal{B}_x^u we have an updated version that has the same reachability function for the agent y .
 - Then we can derive that $\mathcal{B}_{x,y}^u \equiv \mathcal{B}_{x,y}^{u'}$ 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 $u \models \mathbf{B}_x(\mathbf{B}_y(\varphi))$ then $u' \models \mathbf{B}_x(\mathbf{B}_y(\varphi))$.
- (7) Let us consider $y \in \mathbf{U}_a$. Let us assume, without losing generality, that $u \models \mathbf{B}_y(\phi)$. The cases where $u \models \mathbf{B}_y(\neg\phi)$ or $u \models (\neg\mathbf{B}_y(\phi) \wedge \neg\mathbf{B}_y(\neg\phi))$ are similar and therefore omitted.
- Being $y \in \mathbf{U}_a$ we know that the updated version of her/his reachable possibility is $\mathcal{B}_y^{u'} = \{\mathbf{p}' \mid \mathbf{p} \in \mathcal{B}_y^u \wedge \mathbf{p}' = \Psi(\mathbf{a}, \mathbf{p})\}$.
 - Following Definition 3.2 we know that each possibility in $\mathcal{B}_y^{u'}$ 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 $u \models \mathbf{B}_y(\phi)$ iff $u \models \mathbf{B}_y(\phi)$ (similarly if $u \models \mathbf{B}_y(\neg\phi)$ and if $(\neg\mathbf{B}_y(\phi) \wedge \neg\mathbf{B}_y(\neg\phi))$).

□

^aThe 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

untrustworthy fully observant agents in u as $\mathcal{B}_{U_a}^u$ and we recall that, as shown in Appendix A.1, this set corresponds to the possibilities reached by C_{U_a} .

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

(9) The set of the possibilities reachable by *trustworthy fully observant* agents is $\mathcal{B}_{F_a}^u$.

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

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

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

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

- Given that the set $\mathcal{B}_{\mathbf{P}_a, \mathbf{U}_a}^\Psi$ 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 $\forall \mathbf{p} \in \mathcal{B}_{\mathbf{P}_a, \mathbf{U}_a}^\Psi, \mathbf{p} \models \neg\phi$.
- This implies that $\mathcal{B}_{\mathbf{P}_a, \mathbf{U}_a}^\Psi$ reaches only possibilities where the interpretation of ϕ is false and similarly in $\mathcal{B}_{\mathbf{P}_a, \mathbf{U}_a}^\Upsilon$ only possibilities where the interpretation of ϕ is true.
- This means that $\mathcal{B}_{\mathbf{P}_a, \mathbf{U}_a}^\Psi \models \neg\phi$ and $\mathcal{B}_{\mathbf{P}_a, \mathbf{U}_a}^\Upsilon \models \phi$.
- We can then derive that $\mathcal{B}_{\mathbf{P}_a}^\Psi \models \mathbf{C}_{\mathbf{U}_a} \neg\phi$ being $\mathcal{B}_{\mathbf{P}_a, \mathbf{U}_a}^\Psi = \{\mathbf{p} \mid \mathbf{p} \in \bigcup_{\mathbf{q} \in \mathcal{B}_{\mathbf{P}_a}^\Psi} \mathbf{q}(\mathbf{U}_a)\}$ (and similarly $\mathcal{B}_{\mathbf{P}_a}^\Upsilon \models \mathbf{C}_{\mathbf{U}_a} \phi$).
- Finally, being $\mathcal{B}_{\mathbf{P}_a}^{u'} = \mathcal{B}_{\mathbf{P}_a}^\Psi \cup \mathcal{B}_{\mathbf{P}_a}^\Upsilon$ we can conclude that $\mathcal{B}_{\mathbf{P}_a}^{u'} \models \mathbf{C}_{\mathbf{U}_a} \phi \vee \mathbf{C}_{\mathbf{U}_a} \neg\phi$ and therefore $u' \models \mathbf{C}_{\mathbf{P}_a}(\mathbf{C}_{\mathbf{U}_a} \phi \vee \mathbf{C}_{\mathbf{U}_a} \neg\phi)$.

□

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

the negated case).

- Following Appendix A.1 we know that $\mathcal{B}_{\mathbf{F}_a, \mathbf{T}_a, \{j\}}^u$ is equal to the possibilities reached by applying $\mathbf{C}_{\mathbf{F}_a}(\mathbf{C}_{\mathbf{T}_a}(\mathbf{B}_j))$. Given that all these possibilities have the truth value of \mathbf{f} set to 1 it is straightforward to see that $\mathbf{u}' \models \mathbf{C}_{\mathbf{F}_a}(\mathbf{C}_{\mathbf{T}_a}(\mathbf{B}_j(\mathbf{f})))$.
- From the previous items now know that $\mathbf{u}' \models \mathbf{C}_{\mathbf{F}_a}(\mathbf{C}_{\mathbf{T}_a}(\mathbf{f})) \wedge \mathbf{C}_{\mathbf{F}_a}(\mathbf{C}_{\mathbf{T}_a}(\mathbf{B}_j(\mathbf{f})))$ and therefore that $\mathbf{u}' \models \mathbf{C}_{\mathbf{F}_a}(\mathbf{C}_{\mathbf{T}_a}(\mathbf{f} \wedge \mathbf{B}_j(\mathbf{f})))$ as stated in Item (1) (while for the negated case we can easily derive that $\mathbf{u}' \models \mathbf{C}_{\mathbf{F}_a}(\mathbf{C}_{\mathbf{T}_a}(\neg \mathbf{f} \wedge \mathbf{B}_j(\neg \mathbf{f})))$).

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

- First we identify the set of all the possibilities reached by the *fully observant* agents in \mathbf{u} as $\mathcal{B}_{\mathbf{F}_a}^u$.
- 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 $\mathcal{B}_{\mathbf{F}_a, \mathbf{M}_a}^u = \{\mathbf{p} \mid \mathbf{p} \in \mathcal{B}_{\mathbf{M}_a}^q \wedge \mathbf{q} \in \mathcal{B}_{\mathbf{F}_a}^u\}$.
- Now to calculate $\mathcal{B}_{\mathbf{F}_a, \mathbf{M}_a}^{u'}$, following Definition 4.4, we apply $\chi(\mathbf{f}, \mathbf{p}, 0)$ to every element \mathbf{p} of $\mathcal{B}_{\mathbf{F}_a, \mathbf{M}_a}^u$. Let us note that if $e(\mathbf{a}) = 0$, that is if \mathbf{j} **announces** $\neg \mathbf{f}$, we should apply $\chi(\mathbf{f}, \mathbf{p}, 1)$ 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 $\mathcal{B}_{\mathbf{F}_a, \mathbf{M}_a}^{u'} = \{\mathbf{p}' \mid \mathbf{p}'(\mathcal{F}) = ((\mathbf{p}(\mathcal{F}) \setminus \{\mathbf{f}\}) \cup \{\neg \mathbf{f}\}) \wedge \mathbf{p} \in \mathcal{B}_{\mathbf{F}_a, \mathbf{M}_a}^u\}$. It is important to notice that the truth value of the fluent \mathbf{f} in the set of possibilities $\mathcal{B}_{\mathbf{F}_a}^u$ is not important as the application of $\chi(\mathbf{f}, \mathbf{p}, 0)$ on all these possibilities forces their updated version to set the truth value of $\mathbf{f} = 0$ (similarly, for the negated case, the fluent truth value is 1).
- It is then straightforward to see that the set $\mathcal{B}_{\mathbf{F}_a, \mathbf{M}_a}^{u'}$ entails $\neg \mathbf{f}$, as all the reached possibility have the truth value of \mathbf{f} set to 0. Recalling that, as shown in Appendix A.1, the set $\mathcal{B}_{\alpha, \beta}^u$ 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 $\mathbf{u}' \models \mathbf{C}_{\mathbf{F}_a}(\mathbf{C}_{\mathbf{M}_a}(\neg \mathbf{f}))$ (and similarly, in the negated case, $\mathbf{u}' \models \mathbf{C}_{\mathbf{F}_a}(\mathbf{C}_{\mathbf{M}_a}(\mathbf{f}))$).
- Now, to prove that $\mathbf{u}' \models \mathbf{C}_{\mathbf{F}_a}(\mathbf{C}_{\mathbf{M}_a}(\mathbf{B}_j(\neg \mathbf{f})))$ we need to recall that the *mistrustful* agents consider that the announcer \mathbf{j} to be

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

- Following Appendix A.1 we know that $\mathcal{B}_{\mathbf{F}_a, \mathbf{M}_a, \{j\}}^{u'}$ is equal to the possibilities reached by applying $\mathbf{C}_{\mathbf{F}_a}(\mathbf{C}_{\mathbf{M}_a}(\mathbf{B}_j))$. Given that all these possibilities have the truth value of \mathbf{f} set to 0 it is straightforward to see that $u' \models \mathbf{C}_{\mathbf{F}_a}(\mathbf{C}_{\mathbf{M}_a}(\mathbf{B}_j(\neg \mathbf{f})))$.
- From the previous items now know that $u' \models \mathbf{C}_{\mathbf{F}_a}(\mathbf{C}_{\mathbf{M}_a}(\neg \mathbf{f})) \wedge \mathbf{C}_{\mathbf{F}_a}(\mathbf{C}_{\mathbf{M}_a}(\mathbf{B}_j(\neg \mathbf{f})))$ and therefore that $u' \models \mathbf{C}_{\mathbf{F}_a}(\mathbf{C}_{\mathbf{M}_a}(\neg \mathbf{f} \wedge \mathbf{B}_j(\neg \mathbf{f})))$ as stated in Item (1) (while for the negated case we can easily derive that $u' \models \mathbf{C}_{\mathbf{F}_a}(\mathbf{C}_{\mathbf{M}_a}(\mathbf{f} \wedge \mathbf{B}_j(\mathbf{f})))$).

(3) To prove Item (3) let us consider $i \in (\mathbf{S}_a \cup \{j\})$ and that u does entail φ (where $\varphi \in \{\mathbf{B}_i(\ell), \mathbf{B}_i(\neg \ell), (\neg \mathbf{B}_i(\ell) \wedge \neg \mathbf{B}_i(\neg \ell))\}$). The case where $u \not\models \varphi$ 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 \mathbf{S} as the **stubborn** agents do.
- Now, being $i \in (\mathbf{S}_a \cup \{j\})$, we know from Definition 4.4 that the updated version of her/his reachable possibilities is represented by the set $\mathcal{B}_i^{u'} = \{\mathbf{p}' \mid \mathbf{p} \in \mathcal{B}_i^u \wedge \mathbf{p}' = \mathbf{S}(\mathbf{a}, u, \ell, \mathbf{s}), \}$ (The Boolean value \mathbf{s} is either 1, if $i \in \mathbf{S}_a$, or 0, when $i = j$).
- Following Definition 4.4 we know that each possibility in $\mathcal{B}_i^{u'}$ 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 \mathbf{S} from u , when considered from an **stubborn** agent i 's point of view, simply re-applies \mathbf{S} to all the possibilities in \mathcal{B}_i^u . This means that if an agent reached a possibility \mathbf{q} from another possibility \mathbf{p} in u she/he will reach \mathbf{q}' from \mathbf{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 $u \models \varphi$ then $u' \models \varphi$ (similarly if $u \not\models \varphi$ then $u' \not\models \varphi$) with $\varphi \in$

$\{\mathbf{B}_i(\ell), \mathbf{B}_i(\neg\ell), (\neg\mathbf{B}_i(\ell) \wedge \neg\mathbf{B}_i(\neg\ell))\}$ and $i \in (\mathbf{S}_a \cup \{j\})$.

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

- Now to calculate $\mathcal{B}_{\mathbf{P}_a}^{u'}$, following Definition 4.4, we apply $\mathbf{P}(\mathbf{a}, \mathbf{p})$ to every element \mathbf{p} of $\mathcal{B}_{\mathbf{P}_a}^u$. This results in all the possibilities \mathbf{p}' of $\mathcal{B}_{\mathbf{P}_a}^{u'}$ to have the same fluent set of the corresponding possibility $\mathbf{p} \in \mathcal{B}_{\mathbf{P}_a}^u$.
- It is easy to identify two disjoint subsets $\mathcal{B}_{\mathbf{P}_a}^{u'_0}$ and $\mathcal{B}_{\mathbf{P}_a}^{u'_1}$ of $\mathcal{B}_{\mathbf{P}_a}^{u'}$ that contains only possibility such that:
 - $\mathcal{B}_{\mathbf{P}_a}^{u'_0} \not\models \ell$;
 - $\mathcal{B}_{\mathbf{P}_a}^{u'_1} \models \ell$;
 - $(\mathcal{B}_{\mathbf{P}_a}^{u'_0} \cup \mathcal{B}_{\mathbf{P}_a}^{u'_1}) = \mathcal{B}_{\mathbf{P}_a}^{u'}$; and
 - $(\mathcal{B}_{\mathbf{P}_a}^{u'_0} \cap \mathcal{B}_{\mathbf{P}_a}^{u'_1}) = \emptyset$.
- From these two sets, following Definition 4.4 we can now construct the sets $\mathcal{B}_{\mathbf{P}_{a,i}}^{u'_0}$ and $\mathcal{B}_{\mathbf{P}_{a,i}}^{u'_1}$, with $i \in (\mathbf{F}_a \cup \{j\})$, by applying the sub-functions $\chi(\mathbf{f}, \mathbf{p}, 0) \forall \mathbf{p} \in \mathcal{B}_{\mathbf{P}_a}^{u'_0}$ and $\chi(\mathbf{f}, \mathbf{p}, 1) \forall \mathbf{p} \in \mathcal{B}_{\mathbf{P}_a}^{u'_1}$ 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 $\mathcal{B}_{\mathbf{P}_a}^{u'_0}$ and $\mathcal{B}_{\mathbf{P}_a}^{u'_1}$ 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 $\mathcal{B}_{\mathbf{P}_{a,i}}^{u'_0}$ 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 $\forall \mathbf{p} \in \mathcal{B}_{\mathbf{P}_{a,i}}^{u'_0}, \mathbf{p} \not\models \ell$.
- This implies that $\mathcal{B}_{\mathbf{P}_{a,i}}^{u'_0}$ reaches only possibilities where the interpretation of ℓ is false and, similarly, in $\mathcal{B}_{\mathbf{P}_{a,i}}^{u'_1}$ reaches only possibilities where the interpretation of ℓ is true.
- This means that $\mathcal{B}_{\mathbf{P}_{a,i}}^{u'_0} \models \neg\ell$ and $\mathcal{B}_{\mathbf{P}_{a,i}}^{u'_1} \models \ell$.
- It is easy to see, then, that $\mathcal{B}_{\mathbf{P}_a}^{u'_0} \models \mathbf{B}_i(\neg\ell)$ being $\mathcal{B}_{\mathbf{P}_a}^{u'_0} = \{\mathbf{p} \mid \mathbf{p} \in$

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

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

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

- First, to avoid unnecessary clutter let us use i) \mathbf{V}_a to indicate the set of the *observant* agents, *i.e.*, $\mathbf{V}_a = (\mathbf{F}_a \cup \mathbf{P}_a \cup \{j\})$ and; ii) i to indicate a *doubtful* agent, *i.e.*, $i \in \mathbf{D}_a$.
- We then identify the set of all the possibilities reached by the *observant* agents in u as $\mathcal{B}_{\mathbf{V}_a}^u$.
- 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 $\mathcal{B}_{\mathbf{V}_a,i}^u = \{p \mid p \in \mathcal{B}_i^q \wedge q \in \mathcal{B}_i^u\}$.
- Now to calculate $\mathcal{B}_{\mathbf{V}_a,i}^{u'}$, following Definition 4.4, we apply both $\chi(\mathbf{f}, p, 0)$ and $\chi(\mathbf{f}, p, 1)$ to every element p of $\mathcal{B}_{\mathbf{V}_a,i}^u$.
- 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 $\mathcal{B}_{\mathbf{V}_a,i}^{u'_0} = \{p' \mid p'(\mathcal{F}) = ((p(\mathcal{F}) \setminus \{\mathbf{f}\}) \cup \{\neg\mathbf{f}\}) \wedge p \in \mathcal{B}_{\mathbf{V}_a,i}^u\}$ and $\mathcal{B}_{\mathbf{V}_a,i}^{u'_1} = \{p' \mid p'(\mathcal{F}) = ((p(\mathcal{F}) \setminus \{\neg\mathbf{f}\}) \cup \{\mathbf{f}\}) \wedge p \in \mathcal{B}_{\mathbf{V}_a,i}^u\}$. It is important to notice that the truth value of the fluent \mathbf{f} in the set of possibilities $\mathcal{B}_{\mathbf{V}_a}^u$ is not important as the application of $\chi(\mathbf{f}, p, 0/1)$ on all these possibilities forces their updated version to set the truth value of $\mathbf{f} = 0/1$.
- As all the reached possibility from $\mathcal{B}_{\mathbf{V}_a,i}^{u'_0}$ and $\mathcal{B}_{\mathbf{V}_a,i}^{u'_1}$ have the truth value of \mathbf{f} set to 0 and 1 respectively we can easily derive that the former entails $\neg\mathbf{f}$, while the latter entails \mathbf{f} .
- Moreover, being $\mathcal{B}_{\mathbf{V}_a,i}^{u'} = (\mathcal{B}_{\mathbf{V}_a,i}^{u'_0} \cup \mathcal{B}_{\mathbf{V}_a,i}^{u'_1})$, we know that $\mathcal{B}_{\mathbf{V}_a,i}^{u'} \not\models \mathbf{f}$ and $\mathcal{B}_{\mathbf{V}_a,i}^{u'} \not\models \neg\mathbf{f}$. This is true because the subset $\mathcal{B}_{\mathbf{V}_a,i}^{u'_0} \not\models \mathbf{f}$ while $\mathcal{B}_{\mathbf{V}_a,i}^{u'_1} \not\models \neg\mathbf{f}$.
- Recalling that, as shown in Appendix A.1, the set $\mathcal{B}_{\alpha,i}^u$ corresponds to the possibilities reached by $\mathbf{C}_\alpha(\mathbf{B}_i)$, where $\alpha \subseteq D(\mathcal{AG})$ and $i \in D(\mathcal{AG})$,

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

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

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

- When an agent $o \in \mathbf{O}_a$, from Definition 4.4, we know that $\mathbf{p}'(o) = \mathbf{p}(o)$. This means that, independently from the how a possibility \mathbf{p}' has been updated, the point of view any oblivious agent o from \mathbf{p}' is equal to the one that the point of view of o from \mathbf{p} .
- This implies that, $\forall \mathbf{p}' \in \mathcal{B}_i^{u'}$ with $i \in D(\mathcal{AG})$, $\mathbf{p}'(o) = \mathbf{p}(o)$ where $o \in \mathbf{O}_a$.
- This means that for every element in \mathcal{B}_i^u we have an updated version in $\mathcal{B}_i^{u'}$ that has the same reachability function for each oblivious agent o .
- Then, it is easy to see, that $\mathcal{B}_{i,o}^u = \mathcal{B}_{i,o}^{u'}$ 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 , $u \models \mathbf{B}_i(\mathbf{B}_o(\varphi))$ iff $u' \models \mathbf{B}_i(\mathbf{B}_o(\varphi))$.
- Finally, we can conclude that if $u \models \mathbf{B}_i(\mathbf{B}_o(\varphi))$ then $u' \models \mathbf{B}_i(\mathbf{B}_o(\varphi))$.

□

^aThe 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 $\text{pos_w}(T, R, P)$ to identify a generic possibility we will write $\text{pos}(\mathbf{u})$ where the lowercase letter in typewriter font (generally \mathbf{u} , \mathbf{v} or \mathbf{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 \mathbf{f} , that in ASP has the generic form:

$$\text{entails}(T, R, P, F) \text{ :- } \text{time}(T), \text{holds}(T, R, P, F), \text{pos_w}(T, R, P), \text{fluent}(F).$$

will be rewritten as:

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

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

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 $m\mathcal{A}^p$ (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 $m\mathcal{A}^p$. Let a domain D , the belief formulae $\varphi, \varphi_1, \varphi_2 \in D(\mathcal{BF})$, a fluent literal $\mathbf{f} \in D(\mathcal{F})$, an agent $i \in D(\mathcal{AG})$, a group of agents $\alpha \subseteq D(\mathcal{AG})$, and a possibility $\mathbf{u} \in D(\mathcal{S})$ be given. The entailment in $m\mathcal{A}^p$ is defined as follows:

- A. $\mathbf{u} \models \mathbf{f}$ if $\mathbf{u}(\mathbf{f}) = 1$;
- B. $\mathbf{u} \models \mathbf{B}_i(\varphi)$ if for each $\mathbf{v} \in \mathbf{u}(i)$, $\mathbf{v} \models \varphi$;

- C. $u \models \neg\varphi$ if $u \not\models \varphi$;
- D. $u \models \varphi_1 \vee \varphi_2$ if $u \models \varphi_1$ or $u \models \varphi_2$;
- E. $u \models \varphi_1 \wedge \varphi_2$ if $u \models \varphi_1$ and $u \models \varphi_2$;
- F. $u \models \mathbf{E}_\alpha\varphi$ if $u \models \mathbf{B}_i(\varphi)$ for all $i \in \alpha$;
- G. $u \models \mathbf{C}_\alpha\varphi$ if $u \models \mathbf{E}_\alpha^k\varphi$ for every $k \geq 0$, where $\mathbf{E}_\alpha^0\varphi = \varphi$ and $\mathbf{E}_\alpha^{k+1}\varphi = \mathbf{E}_\alpha(\mathbf{E}_\alpha^k\varphi)$.

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 $\forall f \in D(\mathcal{F})$ and $\forall u \in D(\mathcal{S})$ the predicate `holds(u, f)` is true *iff* $u(f) = 1$ while `holds(u, ¬f)` is true *iff* $u(f) = 0$.

- 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. $\nexists u \in \mathcal{U}, u \not\models \varphi$. This can be rewritten as $\forall u \in \mathcal{U}, u \models \varphi$. Hence, for formulae of the type $b(i, \varphi)$ 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 $m\mathcal{A}^p$ and ASP Rules represent the inductive steps of the entailment in $m\mathcal{A}^p$ and PLATO respectively, and it is straightforward to check their correspondence. The base cases are Rule *A* for $m\mathcal{A}^p$ 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 $\bigwedge_{i \in \alpha} \mathbf{B}_i(\varphi)$.
- Rule *G* corresponds to ASP Rule *10*.

9. `not_entails(u, c(α, φ)) :- not entails(v, φ), reaches(u, v, i).`

10. `entails(u, c(α, φ)) :- not not_entails(u, c(α, φ)).`

Similarly to ASP Rule *4* for formulae of the type $c(\alpha, \varphi)$ 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 \mathbf{v} that is *reached* by α .

□

A.5.3 PLATO Initial State Construction Correctness

As already mentioned, the initial state description in $m\mathcal{A}^p$ 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 $i \in D(\mathcal{AG})$ be given. A *finitary S5-theory* is a collection of formulae of the form:

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

$$(iii) \mathbf{C}_{\mathcal{AG}}(\mathbf{B}_i(\phi) \vee \mathbf{B}_i(\neg\phi)) \quad (iv) \mathbf{C}_{\mathcal{AG}}(\neg\mathbf{B}_i(\phi) \wedge \neg\mathbf{B}_i(\neg\phi))$$

Moreover, we require each fluent literal $\mathbf{f} \in D(\mathcal{F})$ 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 $m\mathcal{A}^p$ we will show that PLATO has the same behavior as $m\mathcal{A}^p$ 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.*,

$C_{AG}(\phi)$. 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 $C_{AG}(f)$ or $C_{AG}(\neg f)$. 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 $m\mathcal{A}^p$ 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($C_{AG}(\ell)$), fluent(ℓ).`
12. `initial_dim(2**K) :- K = {fluent(ℓ): unknown_init(ℓ)}.`
13. `pos_w(1..K) :- initial_dim(K).`
14. `holds(u, ℓ) :- init($C_{AG}(\ell)$), pos_w(u), fluent(ℓ).`
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($C_{AG}(\ell)$)` 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 $C_{AG}(B_i(\phi) \vee B_i(\neg\phi))$ are used to filter out the edges of the initial state. In particular, during the initial state construction in $m\mathcal{A}^p$ 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($C(\text{or}(b(i, \phi), b(i, \neg\phi))))$).`
19. `not_b_init(u, v, i) :- pos_w({ u, v }), init($C(\text{or}(b(i, \phi), b(i, \neg\phi))))$).`
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, $\forall \mathbf{u} \in \mathcal{G}, \forall i \in \mathcal{AG}$ we have that $\mathbf{u}(i) = \mathcal{G}$. 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. \square

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 $m\mathcal{A}^p$ 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 $\mathbf{a} \in D(\mathcal{AI})$, a possibility $\mathbf{u} \in D(\mathcal{S})$, and an agent $i \in D(\mathcal{AG})$ be given.

The transition function $\Phi : D(\mathcal{AI}) \times D(\mathcal{S}) \rightarrow D(\mathcal{S}) \cup \{\emptyset\}$ for an executable ontic action¹ is $\Phi(\mathbf{a}, \mathbf{u}) = \mathbf{u}'$, where:

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

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

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

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

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.

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

- Rule H corresponds to:

21. $\text{holds}(u', \ell) \text{ :- causes}(a, \ell), \text{pos_w}(u), \text{pos_w}(u'), \text{plan}(T, a).$

22. $\text{holds}(u', \ell) \text{ :- not causes}(a, \ell), \text{holds}(u, \ell), \text{pos_w}(u), \text{pos_w}(u'),$
 $\text{plan}(T, a).$

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

Similarly, when the action a causes $\neg f$ we will have that $u'(f)$ is equal to 0 (Rule H) while the predicate $\text{holds}(v', \neg f)$ 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 $m\mathcal{A}^p$ as in Rule H the fluents modified are only the ones that belong to the set $e(a, u)$ —namely the effects of a —while the others are preserved from $u(\mathcal{F})$. 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 $m\mathcal{A}^p$ 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. $\text{believes}(u', v, i) \text{ :- believes}(u, v, i), \text{oblivious}(i, a),$
 $\text{pos_w}(\{u, u', v\}).$

As described in Rule I_1 an oblivious agent i , from u' , believes the same set of possibilities \mathcal{U}_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}_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}_i and \mathcal{V}_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. $\text{believes}(u', v', i) :- \text{believes}(u, v, i), \text{fully_obs}(i, a),$
 $\text{pos_w}(\{u, u', v, v'\}).$

This scenario for $m\mathcal{A}^p$ 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}_i of possibilities starting from u and v , believes the updated version of \mathcal{P}_i starting from u' and v' . As shown in the other points the result of both the transition functions on a possibility \mathbf{p} is the same possibility \mathbf{p}' and therefore the updated version of \mathcal{P}_i is equal in both $m\mathcal{A}^p$ 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 action² is $\Phi(\mathbf{a}, \mathbf{u}) = \mathbf{u}'$, where:

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

$$\begin{array}{l} J. \quad \mathbf{u}'(\mathcal{F}) = \mathbf{u}(\mathcal{F}) \\ K. \quad \mathbf{u}'(i) = \begin{cases} \mathbf{u}(i) & \text{if } i \in \mathbf{O}_a \\ \bigcup_{\mathbf{w} \in \mathbf{u}(i)} \Phi(\mathbf{a}, \mathbf{w}) & \text{if } i \in \mathbf{P}_a \\ \bigcup_{\mathbf{w} \in \mathbf{u}(i): e(\mathbf{a}, \mathbf{w}) = e(\mathbf{a}, \mathbf{u})} \Phi(\mathbf{a}, \mathbf{w}) & \text{if } i \in \mathbf{F}_a \end{cases} \end{array}$$

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

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. $\text{holds}(u', \ell) :- \text{plan}(T, a), \text{pos_w}(u), \text{pos_w}(u'), \text{holds}(u, \ell).$

Let us start by showing that the updated possibilities u' and v' , generated from $\Phi(a, u)$ and $\Gamma(a, v)$ respectively, are equal with respect to the fluents truth value. This is easily derived: in fact in $m\mathcal{A}^p$ (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 $m\mathcal{A}^p$ 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. $\text{believes}(u', v, i) :- \text{believes}(u, v, i), \text{oblivious}(i, a),$
 $\text{pos_w}(\{u, u', v\}).$

As for the ontic actions an oblivious agent i , from u' , believes the same set of possibilities \mathcal{U}_i that she believed in u (Rule K_1) and in PLATO i believes, from v' , the set \mathcal{V}_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}_i and \mathcal{V}_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. $\text{believes}(u', v', i) :- \text{believes}(u, v, i), \text{partial_obs}(i, a),$
 $\text{pos_w}(\{u, u', v, v'\}).$

This scenario for $m\mathcal{A}^p$ 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 believed by i in v . This means that a partially observant agent, that necessarily believes the same set \mathcal{P}_i of possibilities starting from u and v , believes the updated version of \mathcal{P}_i starting from u' and v' . As shown in the other points the result of both

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

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

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

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

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

31. $\text{believes}(\mathbf{u}', \mathbf{v}', \mathbf{i}) :- \text{believes}(\mathbf{u}, \mathbf{v}, \mathbf{i}), \text{fully_obs}(\mathbf{i}, \mathbf{a}),$
 $\text{holds}(\{\mathbf{u}, \mathbf{v}\}, \ell), \text{pos_w}(\{\mathbf{u}, \mathbf{u}', \mathbf{v}, \mathbf{v}'\}).$

Given that $\Phi(\mathbf{a}, \mathbf{u})$ 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 Ballarín. 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. Schwarzentruher. 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](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 Zakharyashev. *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 Weerd 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 Weerd, 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 heuristics: 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](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>.